Research Square

Preprints are preliminary reports that have not undergone peer review.
They should not be considered conclusive, used to inform clinical practice,
or referenced by the media as validated information.

Quantum Hash Function Based on Spectral Properties of Graphs and Discrete Walker Dynamics

Mohana Priya Thinesh Kumar
Indian Institute of Technology Dhanbad
Pranavishvar Hariprakash
pranavishvar@gmail.com

Indian Institute of Technology Dhanbad

Research Article

Keywords: Quantum Phase Estimation, Spectral Fingerprinting, Spectral Hashing, Graph Laplacian, Heat Kernel Fingerprinting, Quantum Hash Function, Post-Quantum Cryptography, Discrete Walker Dynamics

Posted Date: June 30th, 2026

DOI: https://doi.org/10.21203/rs.3.rs-9626195/v1

License: This work is licensed under a Creative Commons Attribution 4.0 International License.
Read Full License

Additional Declarations: No competing interests reported.

Quantum Hash Function Based on Spectral Properties of Graphs and Discrete Walker Dynamics

Mohana Priya Thinesh Kumar[1*†] and Pranavishvar Hariprakash[1*†]

[1*]Department of Physics, Indian Institute of Technology (Indian School of Mines), Dhanbad, 826004, Jharkhand, India.

*Corresponding author(s). E-mail(s): manthramohana1@gmail.com; pranavishvar@gmail.com;
†These authors contributed equally to this work.

Abstract

We present *Quantum Graph Hash* (QGH-256), a novel spectral hashing algorithm based on quantum techniques for generating fingerprints from message-induced graphs. Each input message is mapped to a weighted graph using a discrete walker on an $\boldsymbol{n} \times \boldsymbol{n}$ toroidal grid, where edge weights are determined by the message structure. Quantum Phase Estimation (QPE) is then employed to extract spectral information from the graph Laplacian by estimating the phase value. These features capture both structural and dynamical properties of the graph, enabling the method to distinguish between graphs that may share the same eigenvalue spectrum but differ in structure. The resulting spectral fingerprint is compressed into a fixed 256-bit hash, providing a compact representation of the input message. The proposed approach exhibits strong sensitivity to small input changes, a key requirement for cryptographic hash functions. To demonstrate feasibility, we implement QGH-256 on a $\mathbf{4} \times \mathbf{4}$ toroidal grid using Qiskit, and perform simulations with a seeded statevector backend to ensure stable and noise-free results. We evaluate the cryptographic properties of QGH-256 and compare its behavior with current standards. We also discuss its relevance in the context of post-quantum cryptography and outline potential applications of the proposed hashing scheme.

Keywords: Quantum Phase Estimation · Spectral Fingerprinting · Spectral Hashing · Graph Laplacian · Heat Kernel Fingerprinting · Quantum Hash Function · Post-Quantum Cryptography · Discrete Walker Dynamics

1 Introduction

As the digital landscape continues its rapid evolution, ensuring the security and integrity of sensitive data remains a paramount challenge. Cryptographic hash functions, fundamental constructs in modern cryptography since their early formulations, play an indispensable role in data integrity verification by transforming messages of arbitrary length into fixed-length digests While classical standards like MD5, SHA-1, and SHA-256 have historically served as foundational primitives, they increasingly face vulnerabilities, including collision susceptibility and weakened resistance against advanced computational attacks.

The advent of quantum computing introduces an existential threat to these classical cryptographic mechanisms. Algorithms such as Shor's, capable of efficiently factoring large integers, and Grover's, which can accelerate unstructured searches, exploit quantum parallelism to compromise the security assumptions underlying many existing hash functions. These quantum threats necessitate the development of post-quantum and quantum-native cryptographic designs that can withstand both classical and quantum attacks, a need underscored by initiatives like the U.S. National Institute of Standards and Technology Post-Quantum Cryptography (PQC) Standardization process. In response to these critical challenges, we present a novel quantum spectral hashing algorithm designed to generate high-entropy and robust cryptographic fingerprints. Our approach integrates concepts from classical walk theory and quantum spectral graph analysis within a hybrid classical-quantum computational model.

The paper is organized as follows. Section 2 reviews the prerequisite background on classical discrete walks in one and two dimensions. Section 3 introduces the message-induced weighted graph construction that underlies our hashing framework. Section 4 presents the spectral fingerprinting mechanism. Section 5 integrates these components into the complete QGH-256 workflow. Section 6 discusses evaluation of the cryptographic properties of QGH-256. Section 7 examines potential applications of QGH-256. Section 8 discusses quantum threats to classical cryptography and analyzes the compliance of QGH-256 with NIST PQC standards. The paper concludes by highlighting the core contributions of QGH-256, its evaluation using the seeded simulator, and the planned future work.

2 Prerequisites

2.1 One-Dimensional Classical Random Walk

Consider a person standing on a straight line, able to move only in one dimension. The direction of the person's movement is determined by repeatedly tossing a coin. If the outcome of the toss is a head, the person moves one step forward; if the outcome is a tail, the person moves one step backward. After each step, the person tosses the coin again to determine the next move. This simple mechanism forms the core idea of a classical random walk.

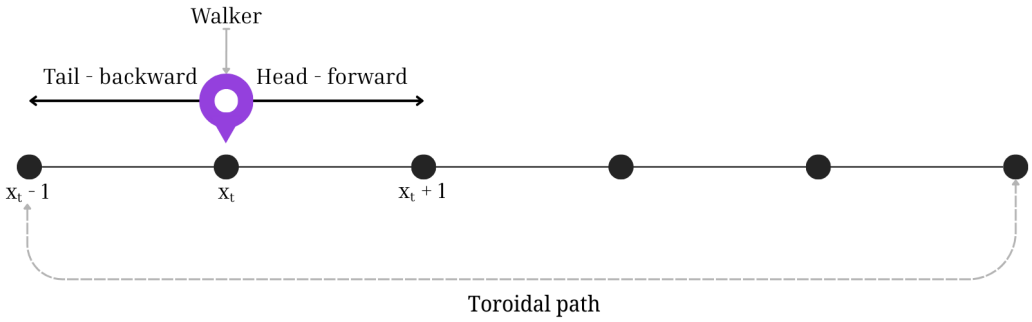


Fig. 1 One-dimensional classical random walk along the toroidal path

The position update rule for each time step can be written as

$$x_{t+1} = \begin{cases} x_t + 1, & \text{if the coin shows head (H),} \\ x_t - 1, & \text{if the coin shows tail (T).} \end{cases}$$

Thus, the motion of the walker is entirely governed by the outcomes of independent coin tosses. Over many steps, this process generates a trajectory consisting of forward and backward movements, producing a random spreading pattern around the starting point.

2.2 Two-Dimensional Classical Random Walk

We now extend the above process to higher dimensions. Consider a person standing on a two-dimensional plane. Unlike the one-dimensional walker that uses a single coin, the two-dimensional walker uses two coins to determine each movement step. The coins are not tossed simultaneously; instead, they are tossed sequentially, and hence the order of outcomes plays a crucial role in determining the direction of motion.

Let the outcome of the first coin be the first bit and the outcome of the second coin be the second bit in a two-bit sequence. Each ordered pair of outcomes corresponds uniquely to one of the four possible directions:

$$\begin{aligned} TT &\rightarrow \text{up (North)}, \\ HH &\rightarrow \text{forward (East)}, \\ HT &\rightarrow \text{down (South)}, \\ TH &\rightarrow \text{backward (West)}. \end{aligned}$$

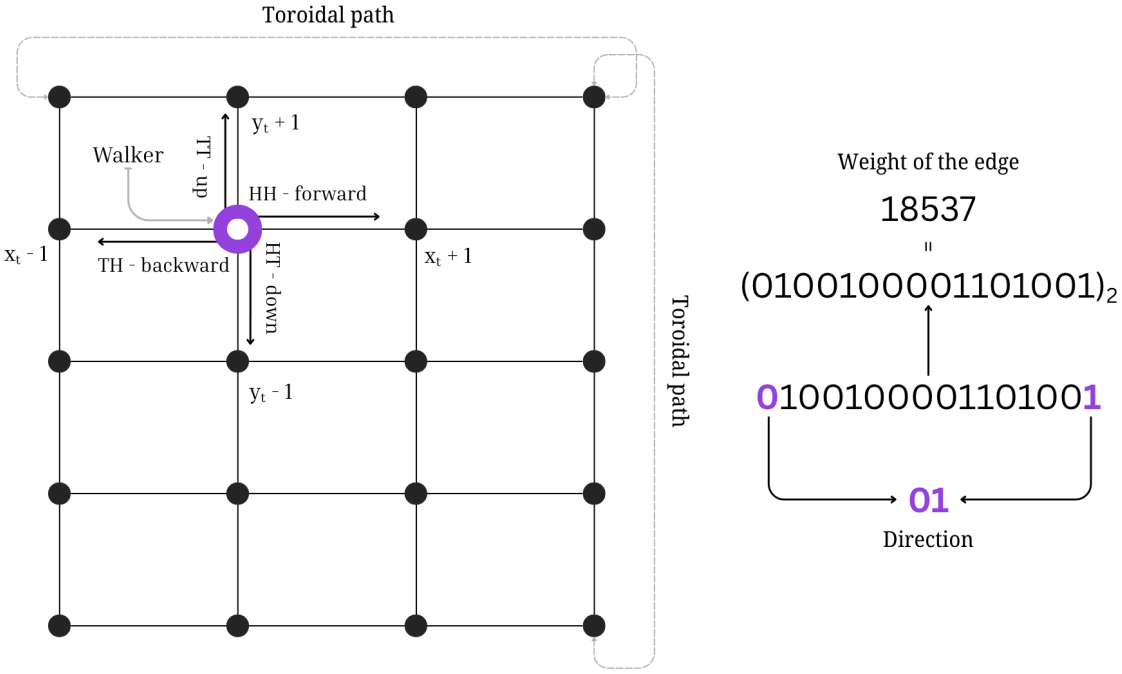


Fig. 2 Two-dimensional classical random walk along the toroidal path

Since both coins are fair and independent, the walker has four equally likely directions to move at each step. Therefore, the movement probabilities are

$$P(\text{North}) = P(\text{South}) = P(\text{East}) = P(\text{West}) = \frac{1}{4}.$$

Based on the two-bit outcome obtained from the sequential coin tosses, the position update rule is

$$(x_{t+1},\, y_{t+1}) = \begin{cases} (x_t,\, y_t + 1), & TT \text{ (move North)}, \\ (x_t,\, y_t - 1), & HT \text{ (move South)}, \\ (x_t + 1,\, y_t), & HH \text{ (move East)}, \\ (x_t - 1,\, y_t), & TH \text{ (move West)}. \end{cases}$$

As in the one-dimensional case, the walker repeats this procedure at every step. Over time, the sequence of ordered coin outcomes generates a path that spreads throughout the plane, producing the classical two-dimensional random walk. The sequential nature of the coin tosses ensures that the order of outcomes uniquely determines the direction, preventing any ambiguity in the movement rule.

3 A Unique Graph For Every Message Using Discrete Walker

Message-Induced Walker Model.

In this work, we propose a procedure that assigns a unique weighted map to every input message. The walker begins on a fixed 4×4 grid, and unlike the classical random walk where movement is chosen randomly, here the walker's direction at each step is entirely determined by the message itself.

3.1 Message Encoding

We first convert the input message into its UTF–8 binary representation. The binary string obtained from the UTF–8 encoding is divided into blocks. If there are not enough bits remaining at the end to form a complete block, we pad the remaining bits with zeros.

3.2 Improvement from the Previous Algorithm

One major issue in the previous approach was that the time required by the walker scaled with the length of the message, since the number of steps increased. This happened because the block size was kept constant ($n = 2$) while the number of steps differed with each message.

In the new approach, we propose to keep the number of steps constant while dividing the block size depending on the message length. This ensures that the number of steps remains fixed regardless of the message size. As a result, the time required for the walk does not scale with the message length. In our case, we fix the number of steps to 10,000.

3.3 Choice of Number of Steps

The choice of the number of steps is important for the efficiency of the algorithm.

If the number of steps is too small (for example, 100), the weight accumulated at each step becomes too large to process efficiently. On the other hand, if the number of steps is too large (for example, 100,000), time efficiency is reduced because the walker must move significantly more across the grid.

Thus, 10,000 steps provides a practical balance between computational efficiency and manageable weight values.

3.4 Direction Encoding and Weight Assignment

The direction of movement is encoded using two bits, similar to the previous method. However, in this case we use the **first and last bits** of each block to determine the direction.

This maps each block to one of the four possible movement directions according to the following correspondence:

$$00 \to \text{down}, \qquad 01 \to \text{up}, \qquad 10 \to \text{right}, \qquad 11 \to \text{left}.$$

This mapping can be modified if required to suit the preferences of the encoder. It can also be made more secure by introducing a keyed mapping between bit pairs and movement directions.

Thus, each step of the classical walker is fixed, while the weights added to each step depends on the message to be encoded. The decimal value obtained from each block is used as the **weight** associated with the edge corresponding to the step taken by the walker.

As the walker continues to trace the path dictated by the message, a weighted graph is formed. The final output is a weighted graph

$$G = (V, E, W),$$

where V is the set of grid nodes, E is the set of edges of the grid, and $W : E \to \mathbb{N}$ assigns to each edge its cumulative traversal count.

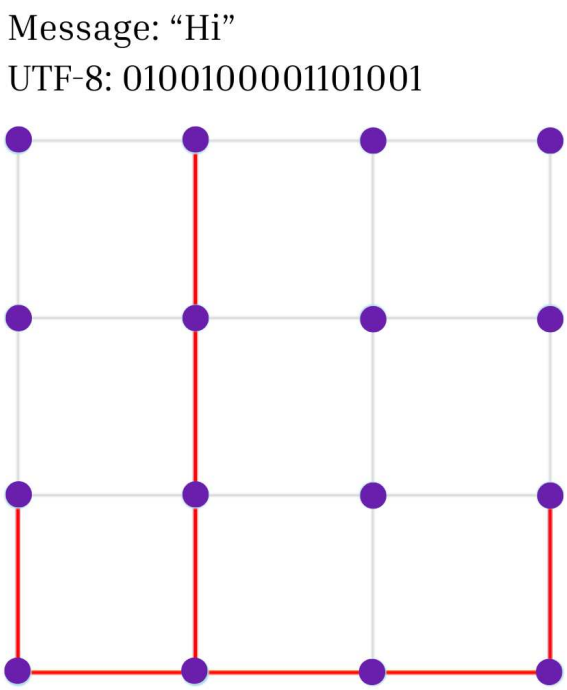


Fig. 3 Message induced discrete walker path traced in 4X4 toroidal grid for the message "Hi"

Each input message produces a distinct and reproducible weighted map: the same message will always generate the same weighted graph.

However, recovering the original message from the resulting graph is computationally hard. Because the graph is undirected, the edge weights record only the total number of times each edge was traversed, without any information about the direction of traversal. Additionally, without knowledge of the walker's final node or the sequence in which edges were visited, reconstructing the exact path becomes extremely difficult. This problem becomes even harder if the mapping between two-bit blocks and movement directions is keyed or randomized in a message-dependent manner through additional processing. Under such conditions, even with full access to the weighted graph, determining the original message is effectively infeasible.

4 Spectral Fingerprint of a Graph

The spectral fingerprint of a graph is a process through which we generate a unique bitstring associated with a given graph using the quantum algorithm known as Quantum Phase Estimation (QPE). Just as every human fingerprint is distinct, the spectral fingerprint of a graph captures its intrinsic structural properties in a way that no two non-isomorphic graphs share the same spectrum under ideal conditions. In this section, we outline the steps required to obtain this spectral representation.

4.1 Hamiltonian Encoding of the Graph Laplacian

We begin by encoding the input graph using its Laplacian matrix L, which incorporates both degree information and connectivity between vertices. The standard Laplacian of a graph is defined as

$$L = D - A,$$

where D is the degree matrix and A is the adjacency matrix. A graph's degree matrix is a diagonal matrix where each diagonal entry is the degree of the corresponding vertex, and all off-diagonal entries are zero. The degree of a vertex is the number of edges connected to it. An adjacency matrix of a weighted graph is a square matrix where each cell (i, j) contains the weight of the edge between vertex i and vertex j.

Example:

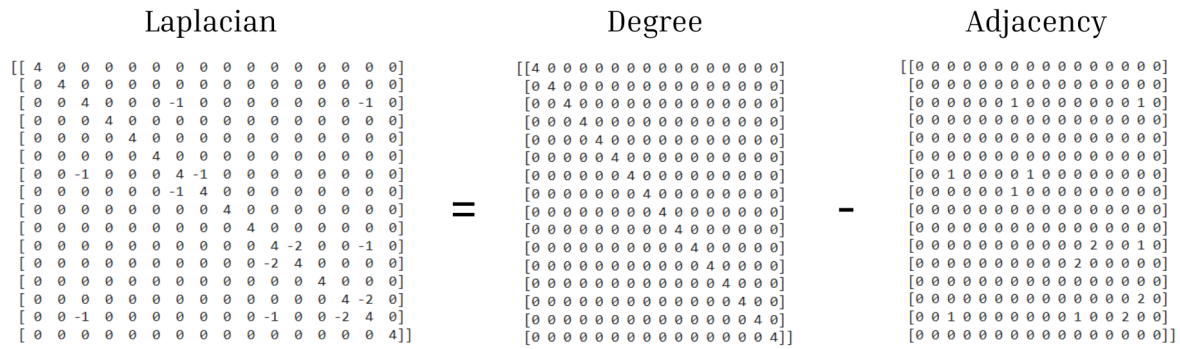


Fig. 4 Graph Laplacian of the weighted graph produced by the message "Hi"

Hermiticity.

A Hermitian matrix is a square complex matrix that is equal to its own conjugate transpose.For a real matrix, being Hermitian is equivalent to being symmetric ($A^T = A$). The Laplacian L of any undirected graph is real and symmetric, and therefore Hermitian:

$$L^\dagger = L.$$

However, L itself is *not* unitary. To use it in quantum phase estimation, we must convert it. This is where Matrix exponentiation comes into play.

4.2 Unitary Transformation via Matrix Exponentiation

To convert the Hermitian Laplacian matrix L (where $L = L^\dagger$) into a Unitary matrix U suitable for quantum computation, we utilize matrix exponentiation. The transformation is defined as:

$$U = e^{iLt}$$

Here, t acts as a time-evolution parameter.

4.2.1 Proof of Unitarity

First, we take the conjugate transpose of the exponential definition:

$$U^\dagger = (e^{iLt})^\dagger = e^{-iL^\dagger t}$$

Since L is Hermitian ($L^\dagger = L$), this simplifies to:

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

Multiplying $U^\dagger$ by U:

$$U^\dagger U = e^{-iLt} \cdot e^{iLt} = e^{(-iLt+iLt)} = e^0 = I$$

Since the product is the Identity matrix I, the operator U is unitary.

4.2.2 Eigenvalue Transformation

A critical property of matrix exponentiation is how it acts on the eigenbasis. While the operation **preserves the eigenvectors**, it **exponentiates the eigenvalues**.

If $|v\rangle$ is an eigenvector of L with a real eigenvalue λ (i.e., $L|v\rangle = \lambda|v\rangle$), we can determine the effect of U using the Taylor series expansion:

$$U|v\rangle = e^{iLt}|v\rangle$$

$$= \left(\sum_{k=0}^{\infty} \frac{(iLt)^k}{k!} \right) |v\rangle$$
$$= \left(\sum_{k=0}^{\infty} \frac{(i\lambda t)^k}{k!} \right) |v\rangle$$
$$= e^{i\lambda t}|v\rangle$$

This step works because $|v\rangle$ is an eigenvector of the operator L, with eigenvalue λ. That is:

$$L|v\rangle = \lambda|v\rangle.$$

Once you know this, every higher power of L acting on $|v\rangle$ becomes:

$$L^2|v\rangle = L(L|v\rangle) = L(\lambda|v\rangle) = \lambda^2|v\rangle,$$

$$L^3|v\rangle = L(\lambda^2|v\rangle) = \lambda^3|v\rangle,$$

and in general:

$$L^k|v\rangle = \lambda^k|v\rangle.$$

so by replacing, we get the above transformation.

Result: The eigenvector $|v\rangle$ remains unchanged, but the eigenvalue λ is mapped to the phase factor $e^{i\lambda t}$.

4.2.3 Trotter-Suzuki Decomposition

For non-commuting matrices A and B, the standard exponential law fails:

$$e^{A+B} \neq e^A e^B$$

This makes direct calculation of the matrix exponential computationally hard. To resolve this, we use the **Trotter-Suzuki decomposition** to approximate the exponential. By slicing the time t into n steps (Trotter steps), we can approximate the total operator as a product of exponentials of the individual terms:

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

As $n \to \infty$, this approximation converges to the exact value, allowing us to implement the simulation using a sequence of simpler quantum gates.

4.3 Quantum Phase Estimation (QPE).

Quantum Phase Estimation aims to estimate the phase ϕ of an eigenvalue of a unitary operator U, defined by the eigenvalue equation:

$$U|v\rangle = e^{2\pi i\phi}|v\rangle$$

where ϕ is a value between 0 and 1. We utilize two registers: an m-qubit "counting" register and the eigenvector $|v\rangle$.

Step 1: Initialization

We begin with the counting register in the all-zero state $|0\rangle^{\otimes m}$ and the target in $|v\rangle$:

$$|\psi_0\rangle = |0\rangle^{\otimes m} \otimes |v\rangle$$

Step 2: Superposition

We apply the Hadamard gate (H) to all m qubits in the counting register. This creates an equal superposition of all computational basis states:

$$|\psi_1\rangle = \left(\frac{1}{\sqrt{2^m}} \sum_{k=0}^{2^m-1} |k\rangle \right) \otimes |v\rangle$$

Or, written in tensor product form for individual qubits:

$$|\psi_1\rangle = \frac{1}{\sqrt{2}} (|0\rangle + |1\rangle) \otimes \cdots \otimes \frac{1}{\sqrt{2}} (|0\rangle + |1\rangle) \otimes |v\rangle$$

Step 3: Controlled-Unitary Operations (Phase Kickback)

We apply a sequence of controlled-U gates. The j-th qubit (from the bottom) controls the unitary U^{2^j}. Due to the **Phase Kickback** phenomenon, applying a controlled-U on eigenvector $|v\rangle$ leaves $|v\rangle$ unchanged but adds a phase $e^{2\pi i\phi}$ to the $|1\rangle$ component of the control qubit.

For the first qubit applying U^{2^0}:

$$\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \xrightarrow{CU^{2^0}} \frac{1}{\sqrt{2}}(|0\rangle + e^{2\pi i\phi \cdot 2^0}|1\rangle)$$

For the k-th qubit applying U^{2^k}:

$$\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \xrightarrow{CU^{2^k}} \frac{1}{\sqrt{2}}(|0\rangle + e^{2\pi i\phi \cdot 2^k}|1\rangle)$$

After applying all controlled gates, the state of the counting register becomes:

$$|\psi_{final}\rangle = \frac{1}{2^{m/2}} \left(|0\rangle + e^{2\pi i(2^{m-1}\phi)}|1\rangle\right) \otimes \cdots \otimes \left(|0\rangle + e^{2\pi i(2^0\phi)}|1\rangle\right) \otimes |v\rangle$$

This can be rewritten mathematically as a summation:

$$|\psi_{final}\rangle = \frac{1}{2^{m/2}} \sum_{k=0}^{2^m-1} e^{2\pi i\phi k}|k\rangle \otimes |v\rangle$$

Step 4: Inverse Quantum Fourier Transform (IQFT)

The QPE state before applying the inverse QFT is:

$$\frac{1}{2^{m/2}} \sum_{k=0}^{2^m-1} e^{2\pi ik\phi} |k\rangle.$$

This matches the definition of the Quantum Fourier Transform of an integer x:

$$\text{QFT}|x\rangle = \frac{1}{\sqrt{2^m}} \sum_{k=0}^{2^m-1} e^{2\pi ixk/2^m} |k\rangle.$$

Compare the phase factors:

$$e^{2\pi ik\phi} \quad \text{vs} \quad e^{2\pi ixk/2^m}.$$

These expressions are equal when

$$\phi = \frac{x}{2^m} \quad \Longrightarrow \quad x = 2^m\phi.$$

Thus, our state is exactly the Fourier transform of the integer $x = 2^m\phi$.

Applying the inverse Quantum Fourier Transform gives:

$$\mathrm{QFT}^\dagger \left(\frac{1}{2^{m/2}} \sum_{k=0}^{2^m-1} e^{2\pi i \phi k} |k\rangle \right) \approx |x\rangle = |2^m \phi\rangle.$$

Step 5: Measurement

We measure the counting register to obtain the integer x. We then calculate the estimated phase:

$$\phi \approx \frac{x}{2^m}$$

Finally, if we used a time parameter t such that $\theta = \lambda t = 2\pi\phi$, we recover the original eigenvalue λ:

$$\lambda = \frac{2\pi\phi}{t}$$

4.4 Why Do We Extract the Phase Values of the Graph Laplacian With Respect to a Superposition State?

In the standard Quantum Phase Estimation (QPE) algorithm, a unitary operator U acts on one of its eigenvectors $|v_k\rangle$, satisfying

$$U|v_k\rangle = \lambda_k |v_k\rangle = e^{2\pi i \phi_k} |v_k\rangle,$$

where $\phi_k \in [0, 1)$ denotes the eigenphase. In this case, the QPE circuit estimates the phase ϕ_k with high probability.

However, in our hashing framework, the input to QPE is not a single eigenvector but a state defined in the node basis of the weighted graph. Each node of the graph corresponds to a computational basis state $\{|0\rangle, |1\rangle, \ldots, |N-1\rangle\}$, and the overall input state can be represented as a superposition over these nodes:

$$|\psi\rangle = \sum_{i=0}^{N-1} \alpha_i |i\rangle.$$

Although we do not input the eigenvectors of the Laplacian directly, each node basis state $|i\rangle$ can itself be expressed as a linear combination of the Laplacian's eigenvectors $\{|v_k\rangle\}$:

$$|i\rangle = \sum_k c_k^{(i)} |v_k\rangle.$$

Thus, the input state $|\psi\rangle$ implicitly becomes a superposition of the Laplacian's eigenvectors.

$$|\psi\rangle = \sum_k c_k |v_k\rangle, \quad \text{where} \quad \sum_k |c_k|^2 = 1.$$

Because QPE operations are linear, they act on each eigencomponent simultaneously. After applying the controlled-U^{2^j} operations, the joint state of the phase and system registers becomes

$$|\Psi\rangle = \frac{1}{2^{m/2}} \sum_{y=0}^{2^m-1} |y\rangle \otimes U^y |\psi\rangle = \frac{1}{2^{m/2}} \sum_k c_k \sum_{y=0}^{2^m-1} e^{2\pi i \phi_k y} |y\rangle \otimes |v_k\rangle,$$

Applying the inverse Quantum Fourier Transform (QFT) we get:

$$|\Psi_{\text{after}}\rangle \approx \sum_k c_k \, |\tilde{\phi}_k\rangle_{\text{phase}} \otimes |v_k\rangle_{\text{system}}.$$

This results in an entangled state between the estimated phase $\tilde{\phi}_k$ and its associated eigenvector $|v_k\rangle$. When the phase register is measured, the state collapses to one of these branches. The probability of obtaining the phase value $\tilde{\phi}_k$ is proportional to $|c_k|^2$, and the system register simultaneously collapses to the corresponding

eigenvector $|v_k\rangle$. Thus, QPE acts as a quantum spectral analyzer, decomposing the input state into its frequency (phase) components:

$$P(\tilde{\phi}_k) \approx |c_k|^2.$$

If we run QPE many times, we see different phase values appear with different frequencies. These frequencies depend on the eigenvalues of U and how much the input state matches each eigenvector. This property is crucial in our hash construction.

How this mechanism prevents Co-Spectral graphs from producing identical fingerprints?

Although two graphs may share the same Laplacian eigenvalue multiset (i.e., they are *co-spectral*), they generally possess different eigenvectors. This is because the QPE sampling distribution depends on the overlaps $|c_k|^2 = |\langle v_k|\psi\rangle|^2$ and not solely on the eigenvalues $\{\lambda_k\}$, co-spectral graphs yield different phase distributions and therefore distinct spectral fingerprints. Our hash is thus sensitive not only to the spectrum but also to the eigenbasis structure, preventing collisions between co-spectral but structurally different graphs.

Example: Representing node as a linear combination of the eigenvectors

Three-Node Weighted Graph

To illustrate how the Quantum Phase Estimation (QPE) algorithm extracts spectral fingerprints from a graph Laplacian, we consider a simple weighted graph G with three nodes $\{v_1, v_2, v_3\}$, where the edge weights encode the connection strengths between nodes.

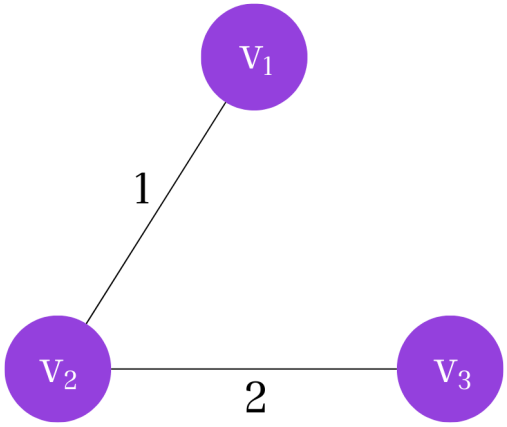


Fig. 5 A simple 3-node weighted graph. Node v_1 is connected to v_2 with weight 1, and v_2 is connected to v_3 with weight 2. There is no direct edge between v_1 and v_3.

The weighted adjacency and degree matrices of this graph are:

$$A = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 2 \\ 0 & 2 & 0 \end{bmatrix}, \quad D = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & 2 \end{bmatrix}.$$

Hence, the graph Laplacian is

$$L = D - A = \begin{bmatrix} 1 & -1 & 0 \\ -1 & 3 & -2 \\ 0 & -2 & 2 \end{bmatrix}.$$

The Laplacian L encodes all structural relationships in the graph: diagonal elements represent node degrees (total connection strengths), while off-diagonal terms represent negative edge weights. The eigenvalues and eigenvectors of L define the graph's intrinsic vibration modes (or diffusion modes).

Node Basis as a Superposition of Eigenvectors

Let us see how the nodes can be represented as a linear combination of the eigen vectors When we physically compute the eigenvalues and eigenvectors of L yields approximately:

$$\lambda_1 = 0.36, \quad \lambda_2 = 2.20, \quad \lambda_3 = 3.44,$$

with corresponding normalized eigenvectors:

$$|\phi_1\rangle = \begin{bmatrix} 0.62 \\ 0.58 \\ 0.53 \end{bmatrix}, \quad |\phi_2\rangle = \begin{bmatrix} 0.77 \\ -0.63 \\ -0.08 \end{bmatrix}, \quad |\phi_3\rangle = \begin{bmatrix} 0.14 \\ 0.51 \\ -0.85 \end{bmatrix}.$$

Each node in the graph corresponds to a computational basis vector. For example, node v_1 corresponds to:

$$|1\rangle = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}.$$

This node-basis state can be expressed as a linear combination of the Laplacian's eigenvectors:

$$|1\rangle = c_1|\phi_1\rangle + c_2|\phi_2\rangle + c_3|\phi_3\rangle,$$

where the coefficients c_k represent the overlap between $|1\rangle$ and each eigenvector:

$$c_k = \langle\phi_k|1\rangle.$$

Using the eigenvectors above, we find:

$$c_1 = 0.62, \quad c_2 = 0.77, \quad c_3 = 0.14.$$

Thus, the node v_1 is not associated with a single eigenmode but rather a weighted superposition of all three Laplacian eigenvectors. We see how when the node basis is used it doesn't relate to a single eigenvector but its superposition.

4.5 Heat trace value

Heat trace value is the exponentiated weighted average of the estimated eigenvalues,

$$h(t) = \sum_k |c_k|^2 e^{-t\tilde{\lambda}_k},$$

where $\{\tilde{\lambda}_k\}$ is the estimated eigenvalue with respect to the superposition state (input to QPE), not the eigenvalues of the graph Laplacian.

Why the Exponential Appears in the Heat Kernel.

The exponential function in the heat kernel is not a convention; it arises directly from the physics of diffusion. The heat (or diffusion) equation on a graph,

$$\frac{\partial f}{\partial t} = -Lf,$$

is a first-order linear time-evolution equation whose generator is the graph Laplacian L. Solving this differential equation yields

$$f(t) = e^{-tL} f(0),$$

so the operator e^{-tL} describes how heat or probability diffuses across the graph over time. When L is diagonalized,

$$Lv_k = \lambda_k v_k,$$

each eigenmode evolves independently as

$$a_k(t) = e^{-\lambda_k t} a_k(0),$$

showing that the eigenvalue λ_k determines the exponential decay rate of the corresponding mode. Thus, the heat trace

$$Z(t) = \mathrm{Tr}(e^{-tL}) = \sum_k e^{-t\lambda_k}$$

captures how all diffusion modes decay over time. The exponential appears naturally because diffusion dynamics are governed by exponential decay of the Laplacian's eigenmodes.

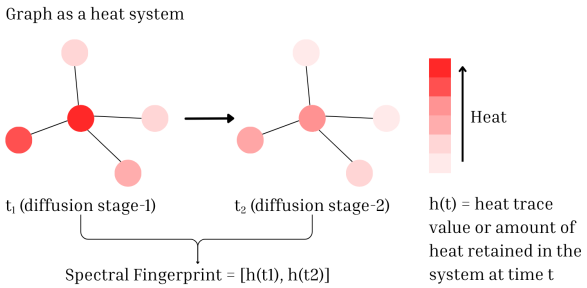


Fig. 6 Physical Intuition Behind Spectral Fingerprint

4.6 Construction of the Spectral Fingerprint

By evaluating the heat trace over multiple time scales $t_1, t_2, \ldots, t_T$, we construct the feature vector

$$\mathbf{h} = [h(t_1), h(t_2), \ldots, h(t_T)],$$

which we define as the spectral fingerprint of the graph. This fingerprint captures both eigenvalue-based and eigenvector-weighted characteristics, providing a unique, multiscale signature of the graph's structure. It remains invariant under graph isomorphism while distinguishing non-isomorphic, cospectral graphs through their differing eigenvector contributions.

5 Workflow - Message to Hash

The overall workflow of the proposed QGH-256 Algorithm is illustrated in Fig. 6. The process consists of several key stages that sequentially transform the input message into a high-entropy spectral fingerprint and finally into a fixed-length binary hash.

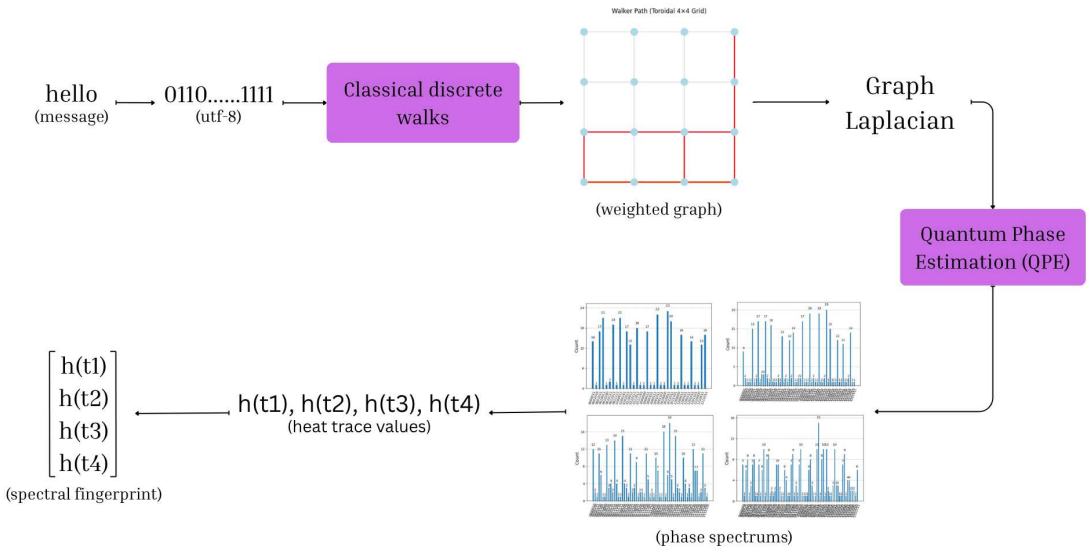


Fig. 7 Overall workflow of the algorithm.

1. **Message Encoding via Discrete Walk:** The input message is first converted into UTF-8 and then mapped onto a weighted graph through classical discrete walker dynamics on a 4×4 toroidal grid.

Each 2-bit symbol (00, 01, 10, 11) in the UTF-8 encoding specifies the walker's direction, producing a message-dependent adjacency structure.

2. **Graph Laplacian Computation:** From the constructed weighted graph, the Laplacian matrix $L = D - A$ is derived, where D is the degree matrix and A is the adjacency matrix. The Laplacian encodes both local connectivity and global structure of the graph.

3. **Quantum Phase Estimation (QPE):** The Laplacian matrix is treated as the Hamiltonian operator for simulated quantum evolution. Using Quantum Phase Estimation, the eigenphase spectrum corresponding to the superposition input state is extracted. This spectrum captures essential graph spectral characteristics.

4. **Heat Trace Evaluation:** For each time evolution step t_i, the exponentiated weighted average of the obtained phase values is computed to derive the *heat trace* value $h(t_i)$. This process is repeated for multiple time steps to track spectral evolution over time.

5. **Spectral Fingerprint Generation:** The sequence of heat trace values $\{h(t_1), h(t_2), \ldots, h(t_n)\}$ forms the spectral fingerprint of the graph. This fingerprint uniquely represents both the structural and dynamical properties of the message-encoded graph.

6. **Hash Construction:** Finally, the spectral fingerprint vector is post-processed and converted into a fixed-length (256-bit) binary hash. This resulting hash serves as a compact, collision-resistant, and quantum-compatible digital signature of the input message.

6 Evaluation of the Cryptographic Properties of QGH-256

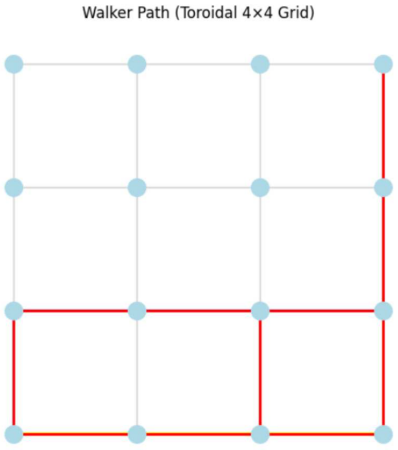
The essential properties that define the strength and reliability of a cryptographic hash function include:

1. Determinism
2. Avalanche Effect
3. Collision Resistance
4. Pre-image Resistance
5. Computation Time

In our proposed quantum hash function based on a message-induced classical random walk and spectral fingerprinting, we validated these core properties using the Seeded Qiskit StatevectorSimulator, which enables noiseless simulation of the underlying quantum circuit. The following sections discuss each property in detail along with the corresponding validation approach.

6.1 A) Determinism

```
sf1= spectral_fingerprint("hello")
hash1 = convert_spectral_fingerprint_into_hash(sf1)
print("Hash:",hash1)
```



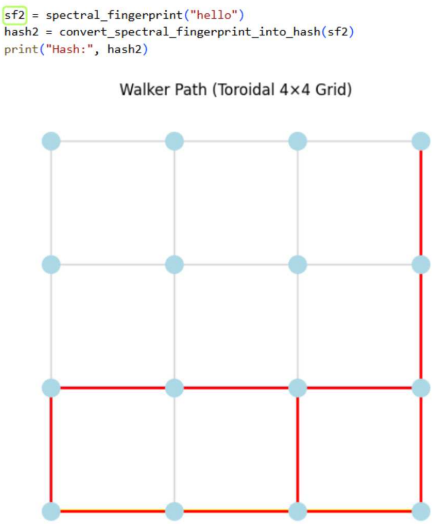


Fig. 8 Code snippet showing same graphs both the times we hash "Hello", proving Deterministic behavior

A hash function must be deterministic, meaning that for a given input message M, the output hash $H(M)$ should always be identical, regardless of how many times the hashing process is executed. Mathematically, this can be expressed as:

$$H(M_1) = H(M_2) \quad \forall\, M_1 = M_2$$

To validate determinism, the same message was hashed twice, and the *Hamming distance* between the resulting bitstrings was calculated as:

$$d_H(x, y) = \sum_{i=1}^{n} |x_i - y_i|$$

where x and y are the two hash outputs, and n is the bit-length of the hash. For a deterministic hash function, $d_H(x, y) = 0$.
In our simulation, identical input messages consistently produced identical spectral fingerprints. The message-induced classical walker generates the same weighted graph structure for the same input, as the Laplacian and hence the eigenvalue spectrum remain in an ideal quantum computer (simulation). Consequently, the spectral fingerprint, which serves as the quantum hash is unique and invariant for a given message, thereby confirming determinism.

On real quantum hardware, however, noise can cause the measured hashes to differ across runs. With improved error-mitigation techniques, larger shot counts, and continued advances in quantum hardware, this variability can be reduced and eventually eliminatedcould be resolved.

6.2 B) Avalanche Effect and C) Collision Resistance

A strong hash function must exhibit the *avalanche effect*, where a minor change in the input message (even a single bit) leads to a significant and unpredictable change in the output hash. This ensures *collision* doesn't occur.

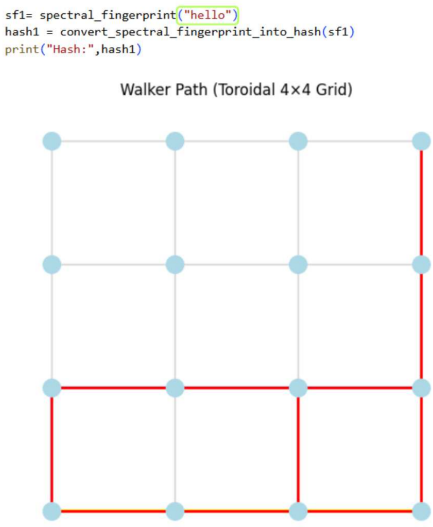


Fig. 9 Graph produced by the message "hello"

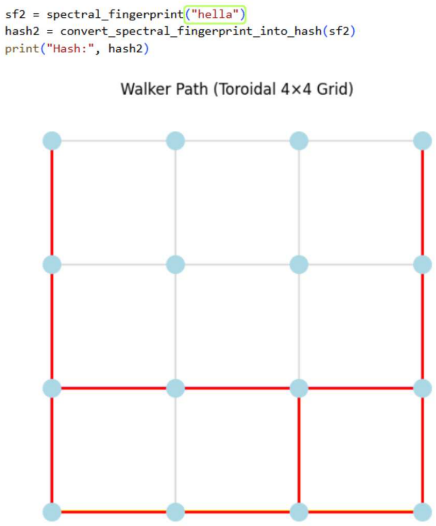


Fig. 10 Graph produced by the message "hella"

To verify this property, we generated hashes for the strings "Hello" and "Hella", differing by one character. The resulting hamming distance was significantly high, confirming the avalanche behavior. At the graph level, the bit difference alters the message-dependent random walk path, leading to a distinct graph and modified edge weights. This, in turn, changes the laplacian substantially, thereby amplifying small input differences into large output deviations.

6.3 D) Pre-image Resistance

Pre-image resistance ensures that given a hash value $H(M)$, it is computationally infeasible to reconstruct the original message M. In the context of our proposed method, the quantum hash is derived from the eigenvalue spectrum of the graph Laplacian, i.e.,

$$H(M) = f(\text{Spec}(L(G_M)))$$

where $\text{Spec}(L(G_M))$ denotes the set of eigenvalues (spectral fingerprint) of the Laplacian L associated with the message-induced graph G_M.

Reconstructing G_M from its spectrum is known as the *graph inverse eigenvalue problem for graphs, problem*, which is computationally expensive to solve in general. Furthermore, even if an approximate reconstruction were possible, the graph being undirected introduces ambiguity as mentioned before in section 3.4, making it hard to infer the specific traversal order of bits corresponding to the input message. Thus, the mapping

from message to graph, and subsequently from graph to spectral fingerprint, is effectively one-way and non-invertible. This ensures strong pre-image resistance.

6.4 E) Computation Time

In our algorithm, the Quantum Phase Estimation (QPE) routine contributes an approximately constant computational cost. This is because the underlying graph is fixed as an $n \times n$ toroidal grid (specifically 4×4 in our implementation). Consequently, the size of the Laplacian matrix and thus the computational cost of the QPE stage remains constant and independent of the message length.

In earlier versions, the classical walker stage scaled with the length of the input message; however, in the revised model, this stage has also been made effectively constant.

As illustrated in the graph below, when processing input messages of varying lengths, the runtime remains nearly constant, with only negligible variations.

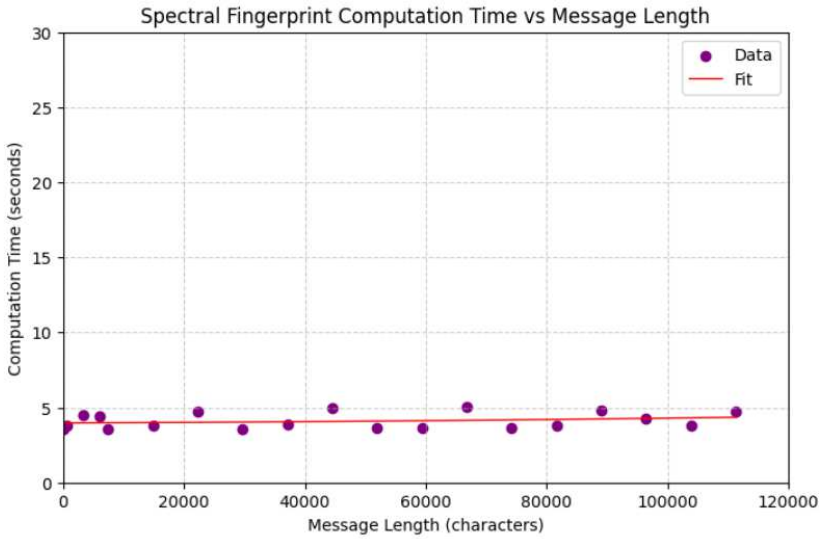


Fig. 11 Graph between message length and computation time.

7 Applications of the Proposed Hash Function

7.1 Proof-of-Work Foundations

Hashcash, proposed by Adam Back (1997), is a **proof-of-work (PoW)** system originally developed to combat spam and denial-of-service attacks. It requires senders to compute a hash satisfying a given difficulty condition (e.g., a fixed number of leading zeros). The computation is intentionally expensive, while verification remains trivial.

In blockchain architectures, Hashcash principles were generalized to regulate block creation. Miners compete to discover a nonce that produces a block hash below a network-defined target, ensuring decentralized consensus and security.

Integration of the Spectral Hash

The proposed **quantum-inspired spectral hashing algorithm** can replace conventional hash functions in PoW systems. Rather than relying on bitwise transformations, it constructs a **graph Laplacian** $L = A - D$ from input data and computes a **spectral fingerprint** via **quantum phase estimation (QPE)**.

The aggregated sequence forms a high-entropy fingerprint that is collision-resistant and quantum-compatible. The mining difficulty is set by defining a target condition for the 256-bit spectral hash, similar to classical proof-of-work, but naturally resistant to quantum speed-ups.

7.2 Data Integrity and Fingerprinting

Data integrity ensures that information remains unaltered during storage or transmission. Conventionally, a cryptographic hash is computed on the data before transfer. The receiver recalculates the hash after download; any mismatch indicates corruption or tampering. A practical example includes software-distribution websites that publish hash values (checksums) for downloadable applications. Users verify authenticity by recomputing the hash locally and comparing it with the official value. Another utilisation is in password protection

Fingerprinting refers to generating a fixed-length digest that uniquely represents larger data. Because cryptographic hashes exhibit the **avalanche effect**, even a one-bit change drastically alters the fingerprint, enabling duplicate detection and authenticity verification.

Our approach provides extreme sensitivity to input perturbations, high entropy. Thus, it functions as both a post-quantum integrity verifier and a high-dimensional data fingerprint.

8 Quantum Threats to Classical Cryptography and Compliance with NIST PQC Standards

8.1 NIST PQC Standards and Requirements

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]. 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 [3], and **Grover's algorithm** for unstructured search [4]. Both have direct implications for the foundational structures of modern cryptography.

The exponential speedup offered by Shor's algorithm threatens widely used **public-key encryption schemes** such as RSA etc [5]. In contrast, Grover's quadratic speedup **reduces the effective security of symmetric-key algorithms**, including AES-128, by effectively halving their key strength [6]. These vulnerabilities could compromise the confidentiality, integrity, and authenticity of sensitive data across financial, governmental, military, and civilian digital infrastructures.

The U.S. National Institute of Standards and Technology (NIST) has initiated the Post-Quantum Cryptography (PQC) Standardization process to identify algorithms resilient against quantum attacks. These standards are designed to replace or augment classical cryptographic primitives with quantum-resistant counterparts.

The NIST PQC framework focuses on four key criteria:

1. **Security**
2. **Performance**
3. **Robustness**
4. **Interoperability**

NIST's objective is to ensure a seamless transition to quantum-safe infrastructure across public and private sectors.

8.2 Compliance of the QGH-256 with NIST PQC Standards

The final step converts this spectral fingerprint into a 256-bit binary hash, representing a high-entropy, collision-resistant identifier of the original data. Unlike traditional hash functions that depend solely on linear bitwise operations, the spectral hash integrates both topological and dynamical information, substantially increasing resistance against both classical and quantum collision or pre-image attacks.

This design achieves compliance with NIST PQC standards as follows:

- **Quantum Resilience:** The spectral fingerprinting process involves complex functions of a graph Laplacian, which introduces non-linear dependencies between input and output. This significantly reduces

vulnerability to Grover-type quadratic search attacks. Building a quantum oracle to compute this reversibly is very complicated and expensive.

- **High Entropy and Collision Resistance:** As mentioned in 6.2, the properties of the graph construction and the Laplacian, ensure the algorithm is resistant to collisions.

- **Interoperability:** The resulting 256-bit output maintains compatibility with existing blockchain and cryptographic infrastructures.

- **Post-Quantum Efficiency:** Since the algorithm is quantum but classically executable through simulations, it provides forward security against near-term quantum threats while remaining efficient on current hardware.

8.2.1 Grover's Algorithm and Its Implications in the proposed hash function

Grover's algorithm is a foundational result in quantum computing that accelerates unstructured search. Given a black-box function $f : \{0, \ldots, N-1\} \to \{0, 1\}$ with a unique marked element, Grover's algorithm can identify the marked input using only $O(\sqrt{N})$ evaluations of f, offering a quadratic speedup over classical brute force search. This is often described metaphorically as "finding a needle in a haystack" using quantum interference: instead of checking each candidate sequentially, the algorithm amplifies the probability amplitude of the correct answer.

A direct consequence is that security assumptions based on exhaustive search, such as preimage resistance of hash functions are weakened. For instance, a classical 2^{128}-step brute-force search can in principle be reduced to approximately 2^{64} steps using Grover's algorithm. Hence, symmetric-key cryptosystems and hash functions are classified as "quantum-vulnerable", requiring increased output sizes or structural countermeasures.

However, Grover's algorithm also incurs nontrivial implementation costs. It requires that the underlying function be represented as a reversible quantum circuit capable of acting on a superposition of all inputs. When the function being attacked is computationally complex, the cost per Grover iteration becomes extremely large, potentially offsetting the quadratic advantage.

Implications for the Proposed Quantum Hash Function.

The algorithm proposed in this paper constitutes a *heavy oracle* from the perspective of Grover's algorithm. The hash function proceeds as follows:

1. Given a message x, construct a weighted graph on a fixed 4×4 toroidal grid.
2. Form the associated graph Laplacian $L = A - D$.
3. Apply Quantum Phase Estimation (QPE)
4. Use these eigenvalues to compute heat traces, which constitute the final hash output.

Several components of this pipeline are intrinsically expensive on quantum hardware:

- **QPE cost:** Quantum Phase Estimation requires multiple ancilla registers and deep controlled-unitary operations to achieve meaningful precision.
- **Matrix exponentiation:** Implementing e^{-iLt} (required for QPE) involves Hamiltonian simulation of a graph Laplacian, which is resource-intensive even for modest matrix sizes.

To apply Grover's algorithm for a preimage attack, an adversary must implement the entire hash function as a reversible quantum circuit. The resulting Grover oracle must:

1. encode all possible messages x in superposition,
2. perform the full toroidal-grid mapping, Laplacian construction, and QPE-based spectral computation coherently on this superposition, and
3. apply a phase flip conditioned on whether the computed spectral hash output equals a target hash value.

Thus, while Grover's algorithm reduces the number of required trials from N to $\sqrt{N}$, each trial becomes substantially more costly because it must simulate a deep quantum circuit implementing the entire spectral hashing pipeline. In effect, the algorithm's internal reliance on QPE and matrix exponentiation transforms it into a computationally heavy oracle, significantly increasing the per-iteration cost in a Grover-based preimage search. Consequently, although Grover's algorithm theoretically threatens any brute-force-based

security assumption, the practical feasibility of such an attack against this quantum-inspired hashing method is constrained by the enormous circuit depth required.

In summary, QGH-256 satisfies the principal objectives outlined by NIST PQC: resistance to quantum attacks, strong entropy characteristics, and practical adaptability within secure communication and blockchain systems.

9 Conclusion

In this work, we introduced a quantum hashing framework using Discrete Classical Walker and Quantum Spectral Fingerprinting. The main outcomes and directions of this study are summarized below:

- **Core Contribution:** We introduce a new quantum hash function that combines a classical discrete walker with quantum spectral fingerprinting to produce a compact, high-entropy hash. The message first influences the path of a classical walker, which in turn shapes a weighted graph. We then apply Quantum Phase Estimation (QPE) to the Laplacian of this graph. Unlike standard QPE methods that operate on a single eigenvector, we input a superposition state so that all eigenvectors contribute to the phase spectrum. This design ensures that even co-spectral but non-isomorphic graphs produce different spectral fingerprints, greatly reducing the chance of collisions and forming the central novelty of our hashing approach.

- **Evaluation on a seeded simulator:** On current Noisy Intermediate-Scale Quantum (NISQ) hardware, algorithms based on Quantum Phase Estimation (QPE) are highly sensitive to noise. Even small amounts of decoherence and gate errors can introduce significant fluctuations in the estimated phase spectrum. In our hashing pipeline, such fluctuations affect determinism, as small variations in the measured phases propagate through the heat-kernel computation and alter the final fingerprint.
  Mitigating these effects on real hardware would require executing QPE with a very large number of measurement shots to obtain stable phase estimates. However, increasing the shot count substantially raises the overall runtime, limiting the practical feasibility of the approach on present-day devices.
  To study the intrinsic behavior of the proposed method, we evaluate our algorithm using a seeded statevector simulator, which provides deterministic and reproducible results by emulating ideal, noiseless quantum evolution. This allows us to analyze the construction without the confounding effects of hardware noise.
  It is worth noting that even well-known quantum algorithms such as Shor's algorithm and Grover's algorithm, while theoretically powerful, face similar practical limitations on current hardware due to noise, decoherence, and limited qubit resources. These constraints necessitate high precision and large numbers of measurements to obtain reliable outcomes.
  Nevertheless, ongoing advances in quantum error correction, qubit stability, and scalable architectures are expected to reduce these limitations. As quantum technology matures, approaches such as the one proposed here may contribute to the development of robust cryptographic primitives in the context of post-quantum security.

10 Declarations

10.1 Funding

No funds, grants, or other support was received.

References

[1] Nielsen, M.A., Chuang, I.L.: Quantum Computation and Quantum Information, 10th anniversary edition edn. Cambridge University Press, Cambridge, UK (2010)

[2] Preskill, J.: Quantum computing in the nisq era and beyond. Quantum **2**, 79 (2018) https://doi.org/10.22331/q-2018-08-06-79

[3] Shor, P.W.: Algorithms for quantum computation: Discrete logarithms and factoring. In: Proceedings of the 35th Annual Symposium on Foundations of Computer Science (FOCS), pp. 124–134 (1994). https://doi.org/10.1109/SFCS.1994.365700 . Also available as arXiv:quant-ph/9508027

[4] Grover, L.K.: A fast quantum mechanical algorithm for database search. In: Proceedings of the 28th Annual ACM Symposium on the Theory of Computing (STOC), pp. 212–219 (1996). Also available as arXiv:quant-ph/9605043

[5] Bernstein, D.J., Buchmann, J., Dahmen, E.: Post-Quantum Cryptography. Springer, Berlin, Heidelberg (2009)

[6] Moody, D., Peralta, R., Smith-Tone, D.: Efficient quantum circuits for aes. Technical Report NIST IR 8109, National Institute of Standards and Technology (NIST) (2016)

[7] Wei, R., Liu, Y., Cui, H., Xie, Y., Zhou, K.: Graph contrastive-and-reconstructive hashing for unsupervised cross-modal retrieval. Data Science and Engineering **10**, 411–427 (2025) https://doi.org/10.1007/s41019-025-00282-1

[8] Martiner, M.: Impact of Quantum Computing on Cryptographic Systems and Blockchain Technology

[9] Weiss, Y., Torralba, A., Fergus, R.: Spectral hashing. In: Advances in Neural Information Processing Systems, vol. 22 (2008)

[10] Liu, W., Mu, C., Kumar, S., Chang, S.-F.: Discrete graph hashing. In: Advances in Neural Information Processing Systems, vol. 27 (2014)

[11] Banerjee, S., Meena, H., Tripathy, S., Panigrahi, P.K.: Fully quantum hash function. arXiv preprint (2024) arXiv:2408.03672 [quant-ph]

[12] Shukla, A., Vedula, P.: Towards practical quantum phase estimation: A modular, scalable, and adaptive approach. arXiv preprint (2025) arXiv:2507.22460 [quant-ph]