A Walk in Quantum Land

Vincent Cai[1], Daniel Jung[1], Jacob Paras[2], Rohan Phanse[3], and Marcus Schubert[4]

Overview

We present a fully quantum hash function based on controlled alternate quantum walks over a cycle graph. The algorithm operates on an 8-qubit system—five qubits encode the walker's position on a 32-node cycle graph, and three qubits represent a 3-dimensional quantum coin. Each input bit dynamically alters the evolution of the quantum walk through input-dependent shift distances governed by a "liveliness" parameter τ, introducing structured, data-driven variability.

At each step, the walker undergoes a unitary transformation that includes a Grover diffusion operator acting on the coin space and a controlled shift operator that propagates the walker based on both coin and message bits. The final state, after processing all message bits, is measured in the computational basis, and the resulting probability distribution is converted into a deterministic bitstring hash using a scaling function.

Key Components

1. Cycle Graph Structure: The algorithm operates on a cycle graph with N=32 vertices (5 qubits). This creates the "space" where the quantum walk occurs.
2. Three-Dimensional Coin: Unlike classical random walks that use a two-sided coin (e.g., heads/tails), this algorithm uses a 3-dimensional quantum coin allowing for three possible movement directions.
3. Liveliness Parameters: The `tau_values` map input bits ("0" and "1") to different "jump" distances (e.g. 0 and 2), creating input-dependent behavior.
4. Grover Diffusion Operator: The coin operator `G = (2/3)*np.ones((coin_dim, coin_dim)) - np.eye(coin_dim)` is a variant of the Grover diffusion operator, which creates quantum interference effects.

How It Works

1. Initial State: The system starts in a well-defined initial state - position 0 with an equal superposition of all three coin states.

[1] Princeton University, Electrical and Computer Engineering
[2] Georgia Institute of Technology, Computer Science and Physics
[3] Yale University, Computer Science
[4] University of Michigan: Ann Arbor, Computer Engineering

2. Message Processing: Each bit in the message controls how the quantum system evolves:
    - The shift operator moves the walker based on the coin state
    - The coin operator creates interference between different paths
    - Each bit in the input causes a different evolution based on the tau value
3. Measurement: After processing all input bits, the final quantum state is measured to generate probabilities across positions.
4. Hash Generation: These probabilities are converted to a bit string using the formula `scaled_p = int((p * 10**l) % (2**s))`.

Why It's Effective

1. Avalanche Effect: Small changes in input (e.g. changing “5” to “6”) produce dramatically different output hash values.
2. Quantum Interference: The algorithm leverages quantum effects that create complex patterns that are extremely difficult to predict without running the full simulation.
3. Sensitivity to Input: Each bit affects the entire quantum state, creating high diffusion.
4. Resistance to Reverse Engineering: Due to the quantum nature of the algorithm, it's computationally difficult to work backward from a hash to find the input.
5. Output Size: The hash length is determined by the parameter N, which is the number of vertices on the cycle. We set N = 32 to achieve a fixed output size of 256 bits, ensuring that our hashing algorithm is one-to-one for inputs up to 256 bits.

The algorithm essentially uses quantum mechanics as a complex mixing function, making it theoretically strong against traditional cryptographic attacks that rely on patterns in classical algorithms.

Criteria

1. Output determinism:

    No stochastic parameters in circuit, and embedding and postprocessing are deterministic processes, thus the entire algorithm is deterministic. We tested this by running 1500 times the same input and always getting the same output, for multiple random inputs.

2. Preservation of entropy:

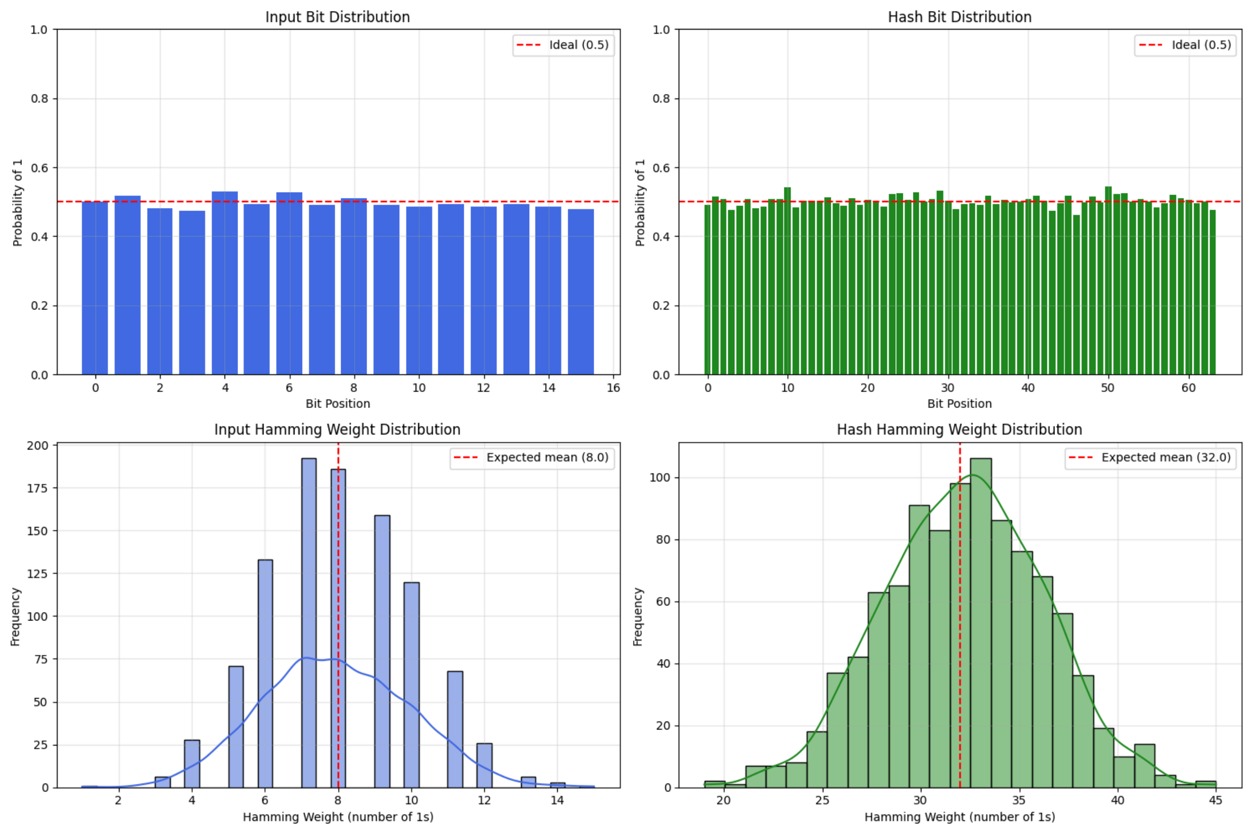

Figure 1. Comparing distributions of input and output bits. We can see the bit positions are uniformly distributed, and the hamming weights follow a normal distribution.

3. <u>Computational difficulty</u>:

   The time complexity of this algorithm is O(n), where n is the number of input bits.

4. <u>Preimage resistance</u>:

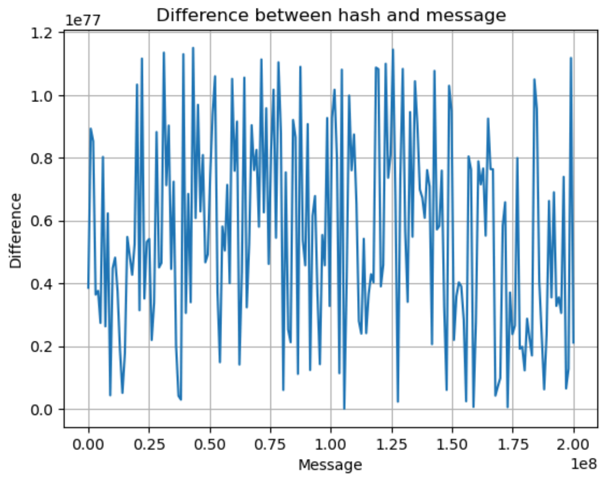

Figure 2. Value of the hash output as the message value changes by a magnitude of 1 per iteration. We can see that small changes in the input result in a wide range of changes in the output.

5. <u>Collision resistance</u>:

   The hashing algorithm experienced zero collisions in an experiment testing it with 2 million different 256-bit values.

6. <u>Computational feasibility</u>:

We use 8 qubits: 5 qubits for position, 2 qubits for the coin operator, 1 cubit for the message

7. <u>Computation time</u>:

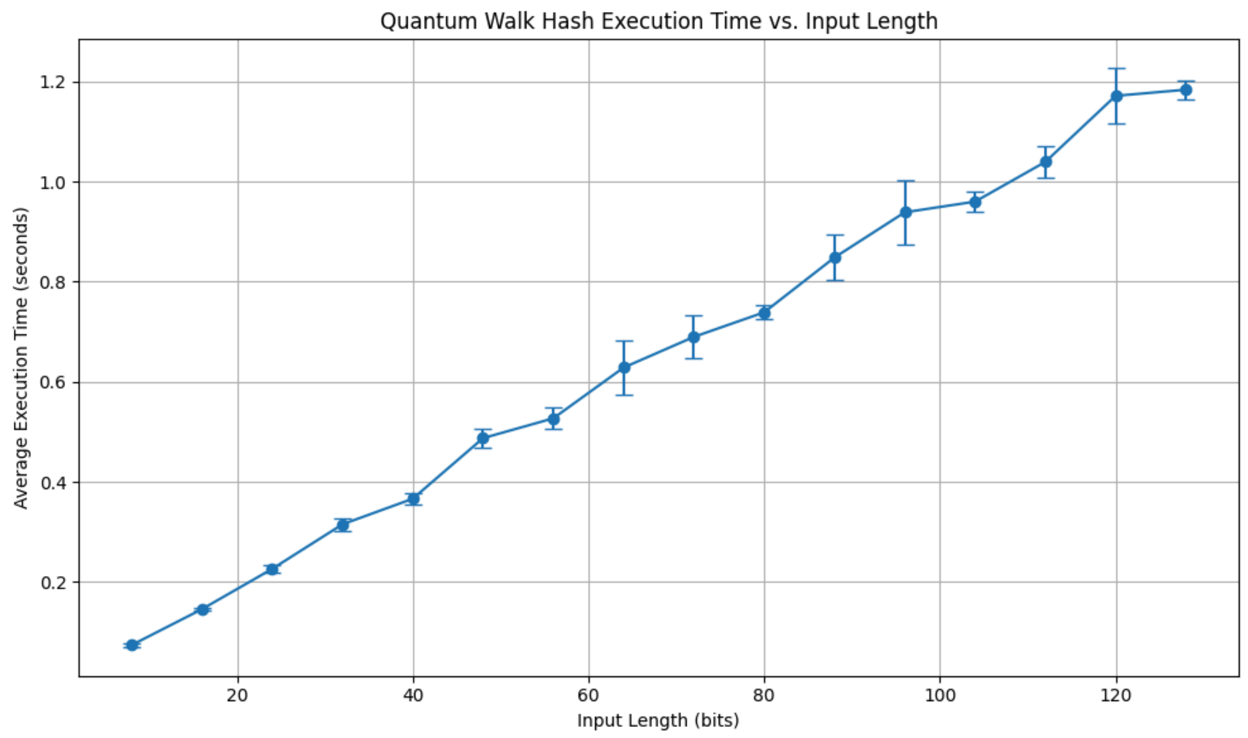

Figure 3. Average execution time increases linearly with the number of bits in the input message.

8. <u>Purely quantum hashing</u>:

This is a quantum analog of the classical random walk, where the Grover coin operator and shift operator are the quantum versions of a "die roll" and step direction in a classical random walk.