A *Newish* Approach to Quantum Hash Function

Exploring Quantum Circuits for Secure Hashing

by - Samradh

The Goal of the challenge:

the students will be required to develop a hash function in Python using their quantum simulator of choice. The input to this function will be an array of 2^N bytes, where N can be any natural number greater than or equal to 5, and the function must produce the byte output of the same size. The function can only utilize the resulting expectation values for each qubit after conducting quantum simulations.

Why it matters:

Quantum hash functions matter because they offer stronger security by leveraging quantum properties like superposition and entanglement. They can resist quantum attacks that break classical hashes, enabling secure communication, authentication, and blockchain applications in the quantum era. They're key to building quantum-safe cryptographic systems for the future.

What is a Quantum Hash Function?

A quantum hash function uses quantum circuits to encode classical input into quantum states and generate a fixed-length output based on measurement or expectation values.

Core Features

- Leverages quantum properties: superposition, entanglement, measurement
- Can offer stronger security guarantees
- Can increase complexity exponentially with circuit size

My Approach

Uses expectation values (Pauli-Z) and parameterized circuits, not measurements, with some other techniques.

Overview of my Approach

- Takes input data of size 2^N bytes
- Produces a hash output of the same size (2^N bytes)
- Uses a parameterized quantum circuit
- Each layer uses rotation gates rx, ry, rz
- Entanglement via cx (CNOT) gates
- ***Output is iteratively rehashed to increase complexity**

*This approach makes it different from most of the currently available quantum hash functions

Architecture of the Quantum Hash Function

- Input data → Convert to angles for parameterized gates
- Build circuit with N qubits, 4 layers of gates
- Run the circuit → obtain statevector
- Compute Pauli-Z expectation values
- Convert to fixed-point → output bytes
- Feed output back as input → repeat until output full

IMP: Hash output = Same size as input

Feedback Loop for Complexity

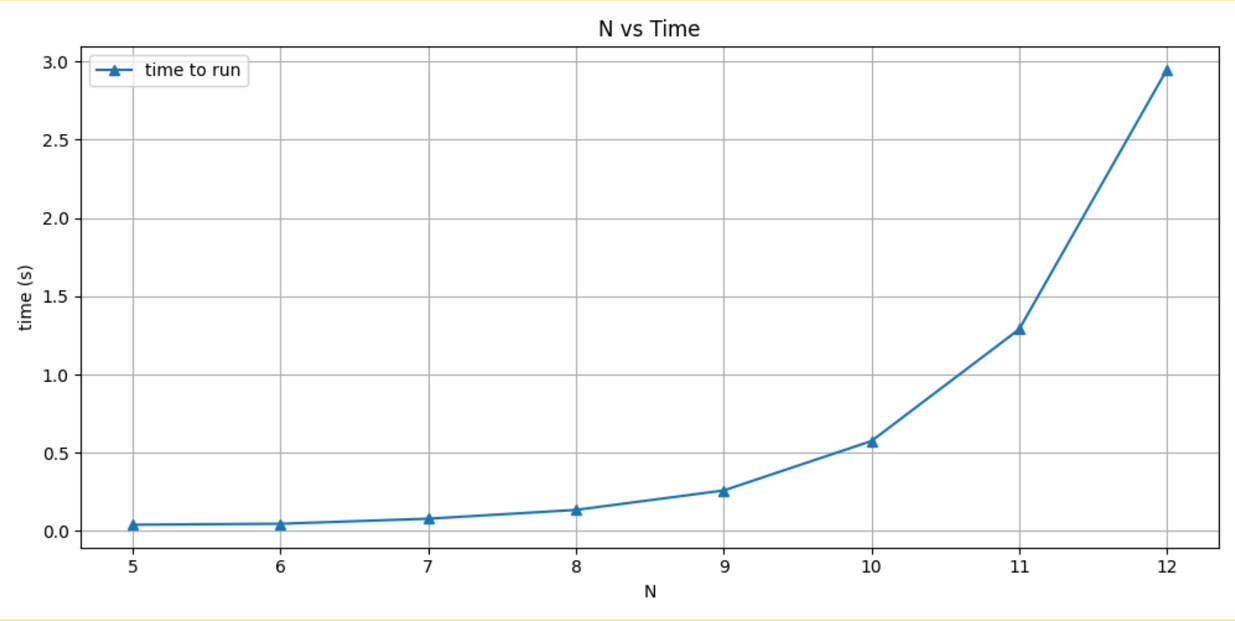
Output of the circuit is re-used as input for next iteration
Increases diffusion and cryptographic complexity
Inspired by multi-round designs in classical cryptography (e.g., Keccak rounds)
Prevents straightforward reverse-engineering

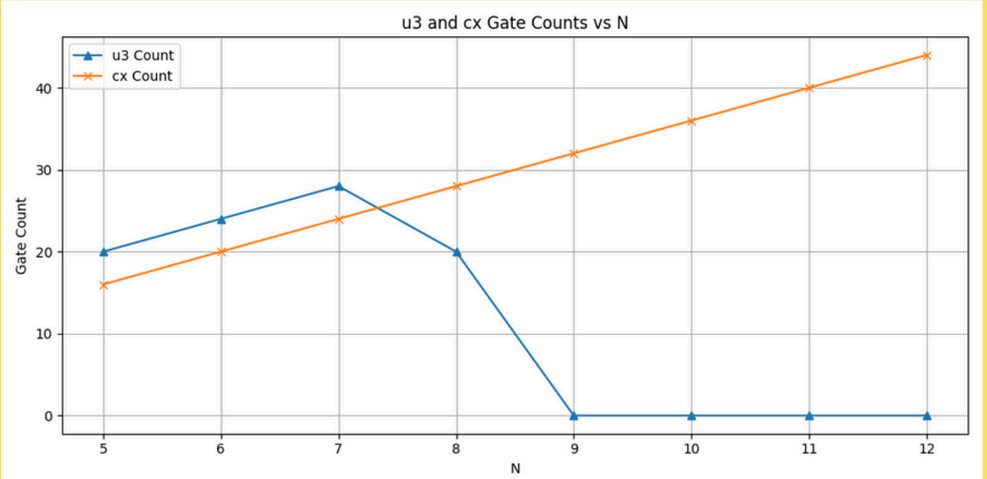
Example

Input → Circuit → Output_1 → Circuit again → Output_2 … until full hash generated

Analysis and Metrics

- **Determinism**: Always same output for same input
- **Entropy**: ~5.65 bits per byte
- **Pre-image Resistance**: No input found for given hash in 1000 trials (N=7)
- **Collision Resistance**: No collisions in 500 trials
- **Computational Complexity**: Time grows exponentially with N





Currently Used Quantum Hash Strategies

1. Quantum Fingerprinting

→ Compares quantum states efficiently

->Not used for secure hashing

2. Phase Estimation-Based Hashing

->Uses eigenvalue mapping to encode data

->More abstract, hard to implement on NISQ devices

3. Grover-Resistant Hashing

->Uses classical hash with large search space

->Only slows down quantum attackers

My Approach:

Combines structured determinism with iterative depth

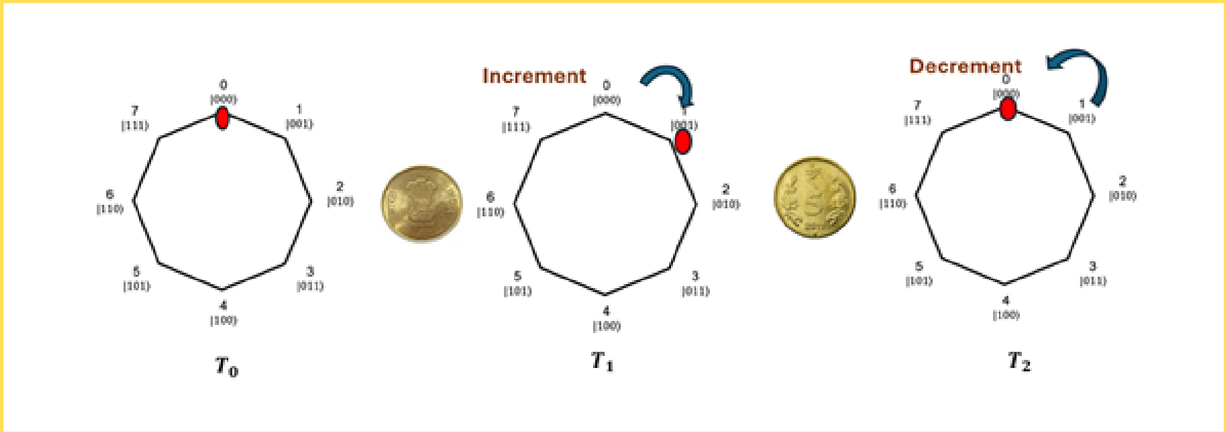
Uses expectation values → interpretable and circuit-friendly

Compatible with NISQ hardware

Some works worth to mention

https://arxiv.org/pdf/2408.03672
https://arxiv.org/pdf/2310.17439
https://www.nature.com/articles/s41598-023-33119-w

While these papers provide really helpful insights into designing hash functions, most of them lack necessary properties required. One of such is Quantum walks, it brings in complexity, but it often produce different hash for same input.



Thank You

The code is more detailed. promise :)