Bruteforcing Bitstreams with Grover's Algorithm

Quantum Computing in Cybersecurity

Background Knowledge

Quantum computing introduces a new paradigm of computation based on qubits, which can exist in superposition and entanglement. Unlike classical bits that are strictly `0` or `1`, qubits can represent multiple states simultaneously. This property allows quantum computers to explore a search space in parallel, offering exponential speedups for certain tasks.

In cybersecurity, one of the main threats posed by quantum computers is their ability to break classical cryptographic systems. Algorithms such as RSA and ECC are vulnerable to Shor's algorithm, while symmetric systems are weakened by Grover's algorithm. In the specific case of brute-forcing bitstreams (e.g., keys or passwords), classical approaches require O(N) time, whereas Grover's algorithm reduces this complexity to O(\sqrt{N})

Bruteforcing
Bruteforcing tries all possible keys until the correct one is found. Classical computers check each option one by one, which is slow for large keys.

Bruteforcing
Classical bruteforce: O(N)

Quantum bruteforce (Grover's algorithm): O(\sqrt{N})

Results and Discussion

3 Qubit Example

For a secret key `101`, the algorithm consistently converged toward the correct result with high probability. Measurement outcomes favored the correct key, demonstrating Grover's theoretical quadratic speedup.

4 Qubit Example

With the secret key `1111`, the success rate dropped noticeably. Out of 1000 attempts, only 183 yielded the correct answer. This highlights a key limitation: Grover's algorithm does not guarantee 100% accuracy. Instead, it increases the probability of success, which must be reinforced by multiple runs.

Precision Loss

As the number of qubits increases, the system becomes more sensitive to noise and hardware limitations. The probability distribution spreads more, leading to ambiguous answers. This is consistent with known challenges in scaling quantum circuits, such as gate fidelity, decoherence, and sampling variance.

Implication for Cybersecurity

Even with precision loss, Grover's algorithm demonstrates a clear advantage over classical brute force. A task that would take 2n2^n2n operations classically can be reduced to roughly 2n/22^{n/2}2n/2 quantum operations, posing a significant risk to symmetric cryptographic systems once fault-tolerant quantum hardware matures.

Methods Used in the Research Paper

This research leverages **Grover's algorithm** implemented in **Qiskit** to simulate quantum brute-force attacks against secret bitstrings.

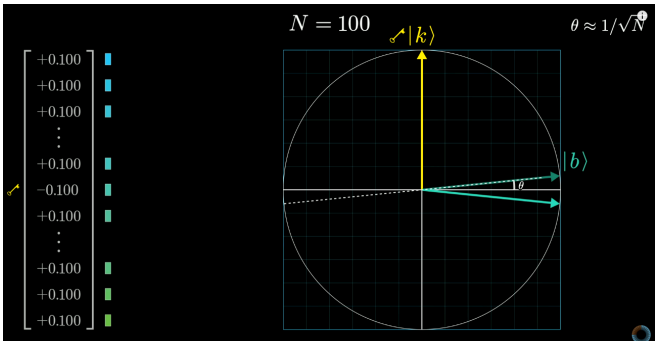
1. **Problem Setup**
    - A secret bitstring is chosen by the user (e.g., `101` or `1111`).
    - The number of qubits in the circuit corresponds to the bitstring length.
2. **Oracle Construction**
    - A **PhaseOracle** is built using the boolean representation of the secret.
    - This oracle marks the correct state by flipping its phase, ensuring it can be amplified without revealing it directly.
3. **Superposition & Amplification**
    - All qubits are initialized into superposition using **Hadamard gates**.
    - Grover's operator (oracle + diffuser) is applied iteratively to rotate the state vector toward the solution.
4. **Execution Modes**
    - **Sampler Mode**: Returns classical counts from multiple runs (shot-based sampling).
    - **Estimator Mode**: Uses expectation values for observables, providing deeper insights into the system's behavior.
5. **Hardware & Noise Considerations**
    - Experiments were conducted on simulators and IBM Quantum backends.
    - Increased qubit counts introduce noise, decoherence, and reduced precision in measurement results.

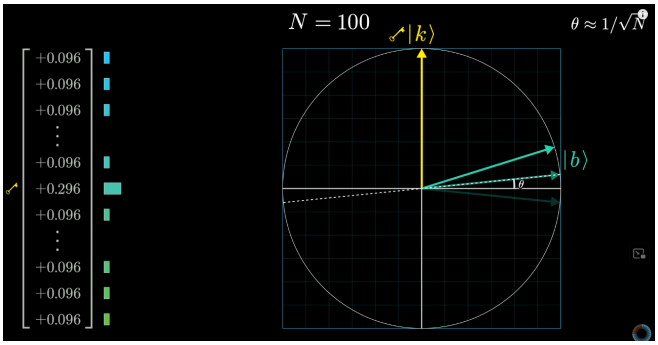
Grover’s Algorithm

In Grover’s algorithm, each possible outcome is represented as a **quantum state vector**.

Only one of these vectors corresponds to the **correct solution**.

We think geometrically to understand Grover's algorithm. If we use a Z gate to rotate the whole state, and repeat this N times we will see that invalid state vector will approach a valid state vector.





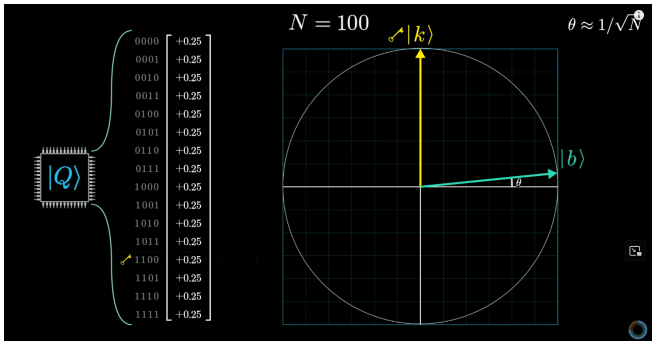


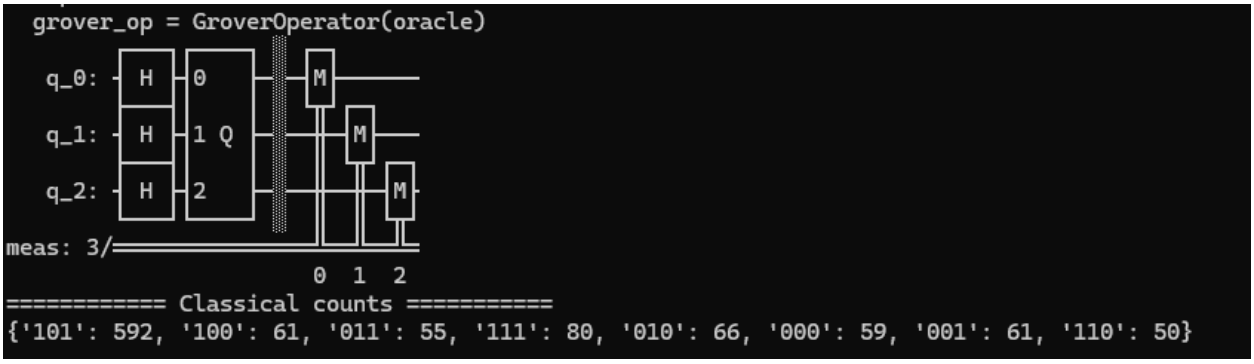
Image Source: Excellent video on quantum computing: 3Blue1Brown, [Quantum Computing]

How it Works

The oracle is a “black box” that flips the phase of the correct quantum state without revealing it (prevents collapse). All qubits are first put into superposition using Hadamard gates. Grover’s algorithm then **amplifies the correct state** through repeated rotations, increasing its probability while suppressing others. Finally, measurement converts the quantum state into classical bits to read the result.

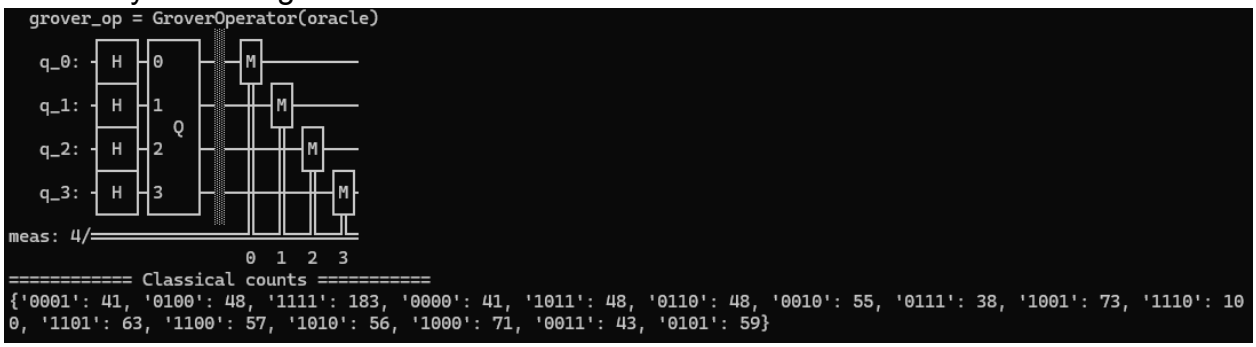
```
# --- Build Grover's Oracle ---
expression = " & ".join([f"x{i}" if bit == "1" else f"~x{i}" for i,
bit in enumerate(SECRET)])
oracle = PhaseOracle(expression)
grover_op = GroverOperator(oracle)


# --- Build Grover's Circuit ---
qc = QuantumCircuit(n_qubits) # we will use 3 qbits in our example
qc.h(range(n_qubits)) # puts all qbits into superposition
qc.compose(grover_op, inplace=True)
if mode == 'sampler':
    qc.measure_all() # CRITIC: Must include this to get classical
counts
```

Result for secret key 101:



Loss of Precision

If we decide to work with 4 qbits, we will start to see more ambiguous answers. I.e., for secret key 1111 we got an answer like:



1000 attempts were made, and only 183 times we measured 1111 as our answer. Comparing to 3 qbit examples we significantly lose our precision. By the nature Grover's algorithm do not guarantee to find the correct answer. Working with more qbuits means more noise, more complex problem to solve and those will cause to lose precision

Conclusion

This work shows how Grover's algorithm can be applied to brute-forcing bitstreams, illustrating both its power and limitations. While small qubit systems achieve strong precision, scaling to larger bit lengths introduces noise and reduces success rates. Despite these challenges, Grover's quadratic speedup makes it a serious threat to modern cryptography, particularly for symmetric systems like AES.

Future work could focus on:

- Error mitigation techniques to reduce noise.
- Running larger-scale experiments on more stable quantum backends.
- Evaluating Grover's algorithm against practical cryptographic key sizes.

The results confirm that while we are not yet at the stage where large-scale brute-forcing with quantum computers is feasible, the trajectory of research suggests it is a matter of *when*, not *if*, quantum attacks will become a practical cybersecurity concern.

Annex

Code also available on my github with setup instructions

```
from qiskit import QuantumCircuit, transpile
from qiskit.circuit.library import PhaseOracle, GroverOperator
from qiskit.quantum_info import SparsePauliOp
from qiskit.transpiler import generate_preset_pass_manager
from qiskit_ibm_runtime import QiskitRuntimeService, EstimatorV2 as
Estimator # api calls
from qiskit_ibm_runtime import SamplerV2 as Sampler
from qiskit.transpiler import generate_preset_pass_manager
from matplotlib import pyplot as plt
# --- User inputs ---
SECRET = input("Enter the secret bitstring (e.g., '11'): ").strip()
n_qubits = len(SECRET)
mode = input("Choose execution mode ('sampler' for classical counts,
'estimator' for expectation values): ").strip().lower()
if mode not in ['sampler', 'estimator']:
    raise ValueError("Invalid mode. Choose 'sampler' or
'estimator'.")
# --- Build Grover's Oracle ---
expression = " & ".join([f"x{i}" if bit == "1" else f"~x{i}" for i,
bit in enumerate(SECRET)])
oracle = PhaseOracle(expression)
grover_op = GroverOperator(oracle)
# --- Build Grover's Circuit ---
qc = QuantumCircuit(n_qubits) # so we go with 2 qbits
qc.h(range(n_qubits)) # puts all qbits into superposition
qc.compose(grover_op, inplace=True)
if mode == 'sampler':
    qc.measure_all() # CRITIC: Must include this to get classical
counts
print(qc.draw())
observables_labels = [f"{'Z'*i + 'I'*(n_qubits-i)}" for i in range(1,
n_qubits+1)]
observables = [SparsePauliOp(label) for label in observables_labels]
service = QiskitRuntimeService()
backend = service.least_busy(simulator=False, operational=True)
if mode == 'sampler':
    qc_transpiled = transpile(qc, backend)
    sampler = Sampler(mode=backend)
    job = sampler.run([qc_transpiled], shots=1024)
```

```
    result = job.result()
    counts = result[0].data.meas.get_counts()
    print("============ Classical counts ===========")
    print(counts)
elif mode == 'estimator':
    # ----- ISA CIRCUIT VERSION -------
    pm = generate_preset_pass_manager(backend=backend,
optimization_level=1) #
    isa_circuit = pm.run(qc)
    print("ISA circuit:")
    print(isa_circuit.draw())
    estimator = Estimator(mode=backend)
    estimator.options.resilience_level = 1
    estimator.options.default_shots = 5000
    mapped_observables = [
        observable.apply_layout(isa_circuit.layout) for observable in
observables
    ]
    job = estimator.run([(isa_circuit, mapped_observables)])
    print(f">>> Job ID: {job.job_id()}")
    job_result = job.result()
    pub_result = job.result()[0]
    values = pub_result.data.evs
    errors = pub_result.data.stds
    plt.plot(observables_labels, values, "-o")
    plt.xlabel("Observables")
    plt.ylabel("Values")
    plt.show()
```

Source: IBM's Qiskit Hello World document used for ISA circuit