Quantum Grover Search for Cracking Caesar Cipher: A Practical Implementation

Jayavarshini.K.S

June 2025

1 Introduction

The Caesar cipher is a classical substitution cipher where each letter in the plaintext is shifted a fixed number of places down the alphabet. It's defined as: Encryption:

$$Enc(Pi, K) = (Pi + K) \mod 16$$

Where:
* Pi is the Plaintext
* k is the secret key (0–25).

2 Ceaser Cipher in Quantum

To implement Caesar cipher logic in a quantum circuit:

- Use **quantum modular addition** to compute p+k.
- Use **Oracle** for phase flip for the marked state .
- Use **uncomputation** (inverse of modular adder) to clean garbage before diffusion.
- Use **Diffusion Operator** to amplify the amplitude of the marked state by reflecting about the mean.

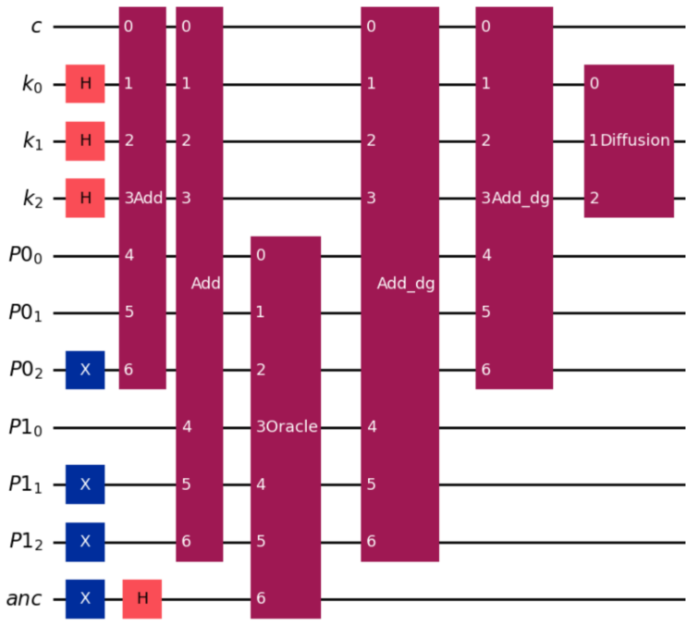

Figure 1: Grover's setting

2.1 Input setup

```
p0 = [0,0,1,0] #plain text
p1 = [1,1,1,1]

c0 = [0,1,0,0] #cipher text
c1 = [0,0,0,1]

n = 4
mu = 1 #No. of solutions(k = [0,0,1,0] = 2)

r = int(np.floor(np.pi/4*np.sqrt(2**(n)/mu)))

c = QuantumRegister(1,'c')
k = QuantumRegister(n,'k')
P0 = QuantumRegister(n,'P0')
P1 = QuantumRegister(n,'P1')
ancilla = QuantumRegister(1,'anc')
# cipher = ClassicalRegister(n,'cipher')
key = ClassicalRegister(n,'key')

qc=QuantumCircuit(c,k,P0,P1,ancilla,key)

for i in range(n):
    qc.x(k[i])
    qc.h(k[i])

for i in range(n):
    if(p0[i]==1):
        qc.x(P0[n-i-1])
    if(p1[i]==1):
        qc.x(P1[n-i-1])

qc.x(ancilla[0])
qc.h(ancilla[0])
```

Listing 1: Input Setting Algorithm

For a key search using Grover's algorithm, we select two elements (P0 and P1), and then we encrypt them with the superposition key K using the ripple-carry addition. First, we set the key and elements of the plaintext to be entered in the oracle. An **ancilla qubit** is prepared by first applying the X-gate and then h-gate so that it becomes :

$$|-\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$$

The Hadamard gate is applied to all qubits of the key K. All values (0 15) exist as probabilities at once (i.e., superposition state). In Figure 6, the X gate is performed before the H gate to optimize the diffusion operator used later. We will describe this in detail later. Second, known elements of plaintext (P0 and P1) are set by applying X gates. Since the initial values of the qubits P0 and P1 are 0, the X gate is applied to the qubit where the plaintext value is 1. Because P0 = 0x2 (i.e. 0b0010) and P1 = 0xF (i.e. 0b1111), the X gate is applied to the second least significant qubit P0(p1) for P0. All qubits of P1 must be 1. The X gate is applied to all qubits for P1(p3 p2 p1 p0).

2.2 Modular Adder

Modular Adder is nothing but the ripple carry adder without the MSB qubit(i.e, z)(Figure 1 and Figure 2)

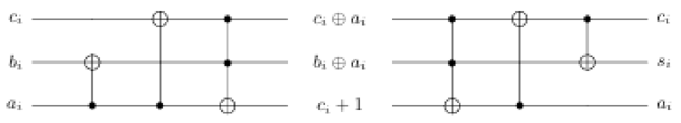

Figure 2: MAJ quantum circuit(left) and UMA circuit(right)

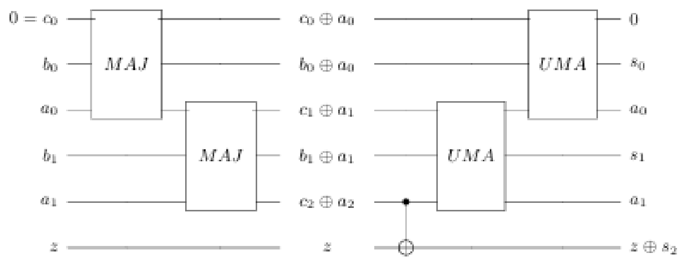

Figure 3: Ripple Carry Adder

The key and the plain texts P0 and P1 are added, and the resulting ciphertext is stored in the plaintext register itself.

2.3 Oracle

```python
def oracle(n, c0, c1):
    c = QuantumRegister(1, 'c')
    k = QuantumRegister(n, 'k')
    P0 = QuantumRegister(n, 'P0')
    P1 = QuantumRegister(n, 'P1')

    ancilla = QuantumRegister(1, 'anc')

    qc = QuantumCircuit(P0, P1, ancilla, name="Oracle")

    # Compute flips
    #Compares the adder result with the input ciphertext
    for i in range(n):
        if c0[i] == 0:
            qc.x(P0[n - i - 1])
        if c1[i] == 0:
            qc.x(P1[n - i - 1])

    #Flipping the ancilla qubit if all P0 and P1 are 1
    qc.mcx(P0[:] + P1[:], ancilla[0])

    # Uncompute flips
    for i in range(n):
        if not c0[i]:
            qc.x(P0[n - i - 1])
        if not c1[i]:
            qc.x(P1[n - i - 1])

    return qc.to_gate()
```
Listing 2: Oracle Function for Grover’s Algorithm

This oracle now takes the input, the added result from the modular adder, and the ancilla qubit. Now the oracle flips the sign only for the states that are equal to the input ciphertexts c0 and c1. This is compared by using the X-gate for the ciphertext qubits that have $|0\rangle$ and followed by a multi-Controlled X gate, where the control qubits are the result of an adder, which produces the ciphertext, and the target qubit is the ancilla qubit.

If all the control qubits are $|1\rangle$ the ancilla qubit, X-gate is applied using the multicontrolled-X gate, then for that state alone phase flip is produced, this is due to the ancilla state $-|-\rangle$.

2.4 Inverse of Modulo Adder

The adder dagger(Figure 3) should be apply before applying diffusion operator because You entangle your **key qubits** with the **plaintext registers** by applying the `adder`. If you don't **undo** this entanglement before the diffuser, then your quantum state looks like:

$$|\Psi\rangle = \sum_k \alpha_k |k\rangle \otimes |P(k)\rangle$$

Where:

- $|k\rangle$ is the key
- $|P(k)\rangle$ is some entangled state of P0, P1, ancilla, etc., dependent on the key

So now each key is **not** in a pure superposition. The state is **not**:

$$\sum_k \alpha_k$$

but rather a sum of **entangled states**.

When you apply `adder_dag`, you **undo the entanglement**: the plaintexts `P0`, `P1` return to their original values. The state becomes:

$$|\Psi'\rangle = \sum_k a_k |k\rangle \otimes |P\rangle$$

Now the diffuser can safely reflect the `a_k` values, which is what it's supposed to do.

2.5 Diffusion Operator

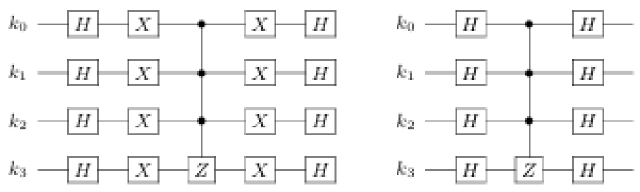

Figure 4: Basic (left) and constant (right) diffusion operators for the Caesar cipher.

The **diffuser** is designed to reflect the current state vector **around the equal superposition mean amplitude**. For `n` qubits, the equal superposition is:

$$|\psi\rangle = \frac{1}{\sqrt{2^n}} \sum_{x\in\{0,1\}^n} |x\rangle$$

And Grover's diffuser applies the transformation:

$$D = 2\,|\psi\rangle\,\langle\psi| - I$$

This means it inverts all amplitudes around the mean. So **the interference works only when the amplitudes are meaningful and comparable** — which requires a clean superposition over just the **search register**

```python
def diffusion(n):
    k = QuantumRegister(n,'k')

    qc = QuantumCircuit(k,name="Diffusion")

    #Not optimised
    qc.h(k[:])
    qc.x(k[:])
    qc.h(k[n-1])
    qc.mcx(k[:n-1], k[n-1])
    qc.h(k[n-1])
    qc.x(k[:])
    qc.h(k[:])
    return qc.to_gate()
```

Listing 3: Basic Diffusion operator Function for Grover's Algorithm

Since the key qubits are applied X-gate in the input setting, a constant number of X gates is used regardless of the number of Grover iterations.

```python
def diffusion(n):
    k = QuantumRegister(n,'k')

    qc = QuantumCircuit(k,name="Diffusion")

    #optimised
    qc.h(k[:])
    qc.h(k[n-1])
    qc.mcx(k[:n-1], k[n-1])
    qc.h(k[n-1])
    qc.h(k[:])

    return qc.to_gate()
```

Listing 4: Constant Diffusion operator Function for Grover's Algorithm

The steps to build the diffusion operator:

$$|\psi\rangle = H^{\otimes n} |0\rangle^{\otimes n}$$

We implement the reflection about $|\psi\rangle$ using **unitary gates**. The idea is:

$$D = H^{\otimes n} \cdot (2|0\rangle\langle 0| - I) \cdot H^{\otimes n}$$

$$2|0\rangle\langle 0| - I = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ 0 & -1 & 0 & \cdots & 0 \\ 0 & 0 & -1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & -1 \end{bmatrix}$$

This is the reflection operator used **in many Grover implementations**, just with a **minus global phase**. Since **the global phase does not have does not have a physical effect** , so:

$$I - 2|0\rangle\langle 0| = \begin{bmatrix} -1 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \end{bmatrix}$$

The amplitude of $|0\rangle$ is **reflected** (sign flipped). All other amplitudes remain **unchanged**. If a quantum state has a component in the direction of $|0\rangle$, that component gets **flipped** in sign.

This can be achieved by applying X-gate for all the key qubits, so that if all the input key qubits are $|0\rangle$ set, then the multicontrolled-Z gate is applied on the last key qubit as target and all other key qubits as controls, if so, the phase of the state is flipped. After this, applying $H^{\otimes n}$ before and after the X-gate and multicontrolled-Z gate gives us the diffusion operator.

$$CZ = HCXH$$

2.6 Grover Iteration

```
mu = 1 #No. of solutions(k = [0,0,1,0] = 2)

r = int(np.floor(np.pi/4*np.sqrt(2**(n)/mu)))

for _ in range(r):
    qc.append(Draper_adder.adder(n),[c[0]]+k[:]+P0[:])
    qc.append(Draper_adder.adder(n),[c[0]]+k[:]+P1[:])

    # Oracle
    qc.append(oracle(n,c0,c1),P0[:]+P1[:]+[ancilla[0]])

    qc.append(Draper_adder.adder_dag(n),[c[0]]+k[:]+P1[:])
    qc.append(Draper_adder.adder_dag(n),[c[0]]+k[:]+P0[:])

    # diffusion operator
    qc.append(diffusion(n),k[:])
```

Listing 5: Grover iteration

The oracle returns the correct key, and the diffusion operator amplifies the amplitude of the returned key. The Grover's search algorithm iterates the oracle and the diffusion operator to sufficiently increase the probability of measuring the correct key and finally measures key qubits. The optimal number of Grover iterations to find a single solution in n qubits ($N = 2^n$)

$$\left\lfloor \frac{\pi}{4}\sqrt{N} \right\rfloor .$$

Since the key is 4 qubits (i.e. n = 4) in proposed designs, the oracle and diffusion operators iterate 3 times and find the key (K=0x2) for the plaintext-ciphertext pair ((PC)=(0xF2,0x14)) with a high probability.

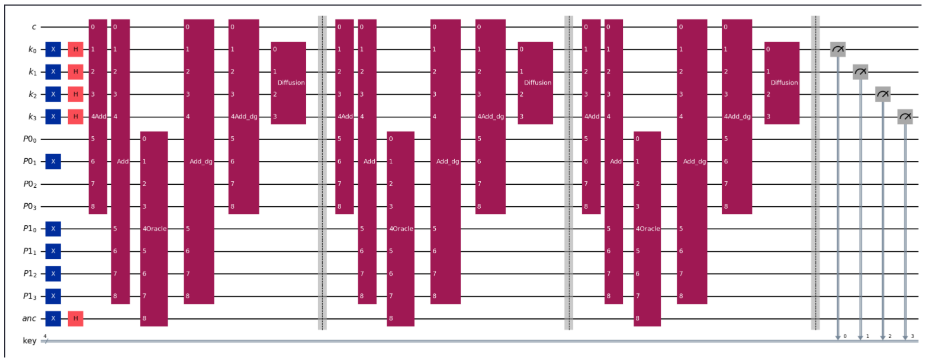

Figure 5: Ceaser cipher circuit

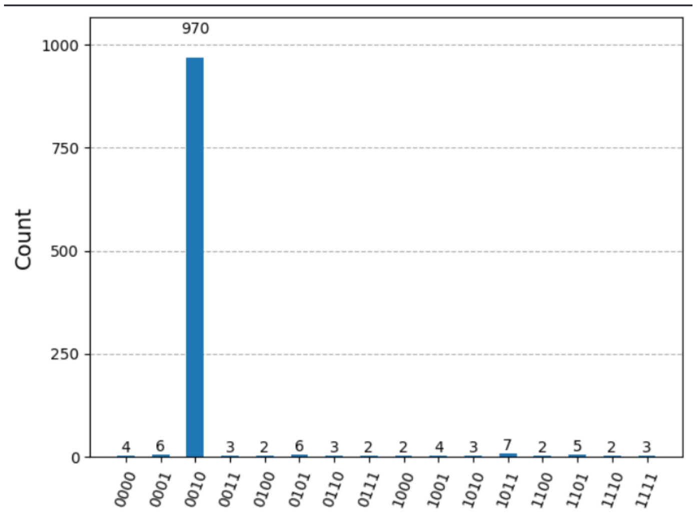

Figure 6: Histogram of output from the simulator

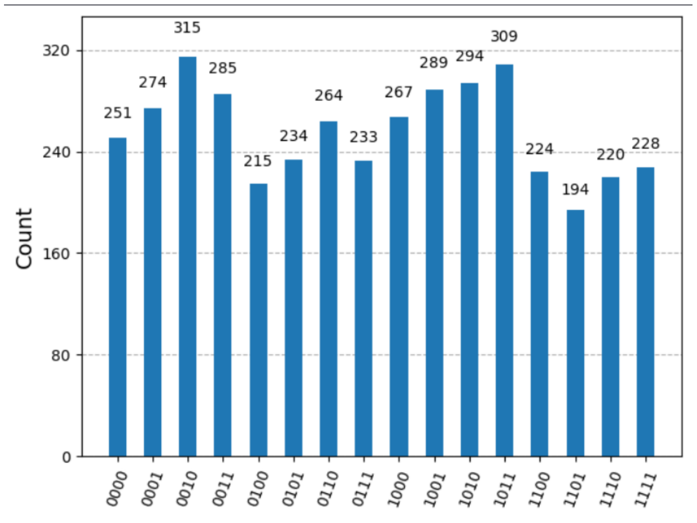

Figure 7: Histogram of output from ibm brisbane