Information Security in the Quantum Era

Threats to modern cryptography: Grover's algorithm

Mihai-Zicu Mina[1] and Emil Simion[2]

[1]*Faculty of Automatic Control and Computers*
[2]*Faculty of Applied Sciences*
University POLITEHNICA of Bucharest, 060042 Bucharest, Romania

mihai_zicu.mina@stud.acs.upb.ro emil.simion@upb.ro

Abstract

Information security plays a major role in the dynamics of today's interconnected world. Despite the successful implementation and effectiveness of modern cryptographic techniques, their inherent limitations can be exploited by quantum computers. In this article we discuss Grover's quantum searching algorithm and its impact on the security of modern symmetric ciphers. More specifically, we present its formal description and give an implementation of the algorithm using IBM's Qiskit framework, which allows us to simulate and run the program on a real device.

Keywords: symmetric cipher, quantum computer, Grover's algorithm

1 Context and motivation

The entire global security infrastructure essentially relies on a combination of the public-key cryptography model envisioned in late 1970s to perform key distribution and the use of fast symmetric ciphers to perform encryption [1–4]. RSA is ubiquitous in such context, but its effectiveness and security only rely on a computational hardness assumption. In other words, powerful enough computational devices could pose a threat. As it turns out, quantum computers started to receive much attention in part due to this particular aspect. Shor's algorithm [5] could break RSA in a timely manner, which is something we can definitely not say about even the most powerful supercomputer we have today. For now though, existing quantum computers are not advanced enough to run Shor's algorithm for those large parameters used by RSA, however the threat is real, inevitable and hence must be addressed.

Symmetric ciphers such as AES suffer from being vulnerable to some degree to Grover's algorithm [6], a fundamental result in the quantum information field that can speed up brute-force attacks, reducing the security of the key to half its length, meaning 256-bit keys offer the security of 128-bit keys against a quantum computer running Grover's algorithm [7–9]. Here we focus on providing a detailed description of how Grover's algorithm operates and we analyze some results for an implementation of it on IBM's Quantum Experience platform, with source code given in Listing 1.

2 Grover's algorithm

2.1 General description

In 1996, Lov Grover devised an algorithmic procedure that uses the principles of quantum computation to search for an element in an unstructured database [6]. The algorithm bears his name and it offers a quadratic speedup over classical methods for the same task. Thus, a direct application of the algorithm is searching for symmetric keys in key spaces, which are essentially unstructured databases. Since AES is pretty much vulnerable to brute-force attacks only, this is exactly how Grover's algorithm threatens it. Considering an n-bit AES key, the size of the key space is $N = 2^n$. Classically, we need $N/2$ iterations on average to find the desired key, but we only need roughly $\sqrt{N} = 2^{n/2}$ iterations using Grover's algorithm, effectively reducing the security of the key to $n/2$ bits in a quantum scenario. Grover's algorithm makes the entry we are looking for more likely to be found than any other one from the entire search space. An oracle is used to "mark" the desired solution, followed by several iterative transformations that aim to amplify the probability associated with the correct answer.

Before proceeding to the detailed analysis of the algorithm, let's have a closer look at what we are trying to achieve, from a non-quantum perspective [10]. We consider a binary function that takes an n-bit string as input and outputs either 0 or 1. Assuming we are given the function as a oracle and the output is always 0 except for one value, we are asked to find the solution for which the output is non-zero.

$$f\colon \{0,1\}^n \to \{0,1\}, \qquad f(\mathbf{x}) = 0, \quad \forall\, \mathbf{x} \neq \mathbf{x}^*$$

Given this context, all we can do is randomly choose a value from the set of bit strings and query the oracle for the output. There are $N = 2^n$ elements, making our guess correct with a probability of $1/N$. For simplicity, the domain of the function is labeled X and our first guess is element $\mathbf{x_1}$. When $f(\mathbf{x}_1) = 1$, we got really lucky and the problem is solved with one query. Most likely, though, we are not done so fast and we must take another guess $\mathbf{x_2}$. After this first query, the probability of having found the solution is

$$\Pr(\mathbf{x}_1 = \mathbf{x}^*) + \Pr(\mathbf{x}_2 = \mathbf{x}^*) \equiv p_1 = \frac{2}{N}, \qquad \mathbf{x}_1 \in X, \quad \mathbf{x}_2 \in X \setminus \{\mathbf{x}_1\}.$$

Next, we query again for $\mathbf{x}_2$ and in case it is still not the solution, we choose $\mathbf{x}_3$. After this step, the probability associated with the solution is

$$p_2 = p_1 + \frac{1}{N} = \frac{3}{N}, \qquad \mathbf{x}_3 \in X \setminus \{\mathbf{x}_1, \mathbf{x}_2\}.$$

Following this line of reasoning and taking into consideration the worst case scenario, the solution is found after we have swept through all the other values. Generally, we notice that the probability of our guess being the solution increases after each query. Indeed, after $k = N-1$ queries, the probability becomes 1.

$$p_k = \frac{k+1}{N}, \quad \mathbf{x}_{k+1} \in X \setminus \{\mathbf{x}_1, \ldots, \mathbf{x}_k\}$$

Grover's algorithm operates on a somewhat similar principle, but requiring a significantly lower number of queries to the oracle in order to find the solution with high probability. The implementation of the oracle is given by a multi-qubit gate of the following form:

$$U_f\colon\ |\mathbf{x}\rangle\,|y\rangle \mapsto |\mathbf{x}\rangle\,|y \oplus f(\mathbf{x})\rangle\,, \quad \mathbf{x} \in X,\ y \in \{0,1\}.$$

We can set the n-qubit first register to an equally weighted superposition of all states in the computational basis, while keeping the second register initialized in $|0\rangle$.

$$U_f \left|+\right\rangle^{\otimes n} |0\rangle = U_f \left(\frac{1}{\sqrt{N}} \sum_{\mathbf{x}\in X} |\mathbf{x}\rangle \right) |0\rangle = \frac{1}{\sqrt{N}} \sum_{\mathbf{x}\in X} |\mathbf{x}\rangle \, |f(\mathbf{x})\rangle \equiv |\Psi\rangle$$

Given how the function is defined, we can split the expression to emphasize the solution:

$$|\Psi\rangle = \frac{1}{\sqrt{N}} |\mathbf{x}^*\rangle \, |1\rangle + \frac{N-1}{\sqrt{N}} \sum_{\mathbf{x}\neq\mathbf{x}^*} |\mathbf{x}\rangle \, |0\rangle \, .$$

A straightforward measurement at this point will return the solution with a probability of $1/2^n$, which is no better than the classical approach. The trick is to find a way to manipulate the state of the quantum register in order to have a higher probability associated with the solution state. One first step is to use the *phase kickback* idea and notice that we can exclude the target qubit from our discussion, as the transformation leaves it intact.

$$U_f \, |\mathbf{x}\rangle \, |-\rangle = (-1)^{f(\mathbf{x})} \, |\mathbf{x}\rangle \, |-\rangle \quad \Longrightarrow \quad U_f \colon \, |\mathbf{x}\rangle \mapsto (-1)^{f(\mathbf{x})} \, |\mathbf{x}\rangle$$

The oracle has been redefined to operate on the data register only. We notice that the element for which we are searching is now identified by having its state get a phase shift.

Setting aside the initialization and measurement phases, the algorithm basically consists of a successive application of a transformation, which boosts the probability amplitude of the solution state and diminishes those of the other states in the superposition. The iteration first queries the oracle, marking the solution with a global phase. This makes the probability amplitude become negative, which is then "reflected" about the mean value of all probability amplitudes, making it positive again, but more importantly, larger. We need to define another transformation before being able to describe how this "reflection" happens. For this, let's first label the uniform superposition of the n-qubit data register,

$$|\Psi\rangle \equiv \left|+\right\rangle^{\otimes n} = H^{\otimes n} \, |0\rangle^{\otimes n} \, .$$

Now, we consider the transformation that induces a global phase of π radians on all states of the register orthogonal to $|00\ldots0\rangle$,

$$U_0 \colon \, |0\rangle^{\otimes n} \mapsto |0\rangle^{\otimes n} \, , \quad |\mathbf{x}\rangle \mapsto - \, |\mathbf{x}\rangle \, , \quad |\mathbf{x}\rangle \neq |0\rangle^{\otimes n} \, .$$

The following transformations are known as *Grover diffusion* and *Grover iteration*, respectively:

$$U_\Psi \equiv H^{\otimes n} U_0 H^{\otimes n}, \qquad G \equiv U_\Psi U_f .$$

The complete quantum circuit is depicted in Figure 1. Each pair of gates enclosed in dashed lines is the Grover iteration, which is applied $\mathcal{O}(\sqrt{N})$ times. Finally, a measurement on the first n qubits will return the solution with a probability close to unity. The last qubit is the ancillary state, such that a query to the oracle will mark the solution by inverting the sign of the probability amplitude. A simplified and more compact version of the circuit is shown in Figure 2, where the last qubit is omitted and the overall Grover iteration is represented as a single gate.

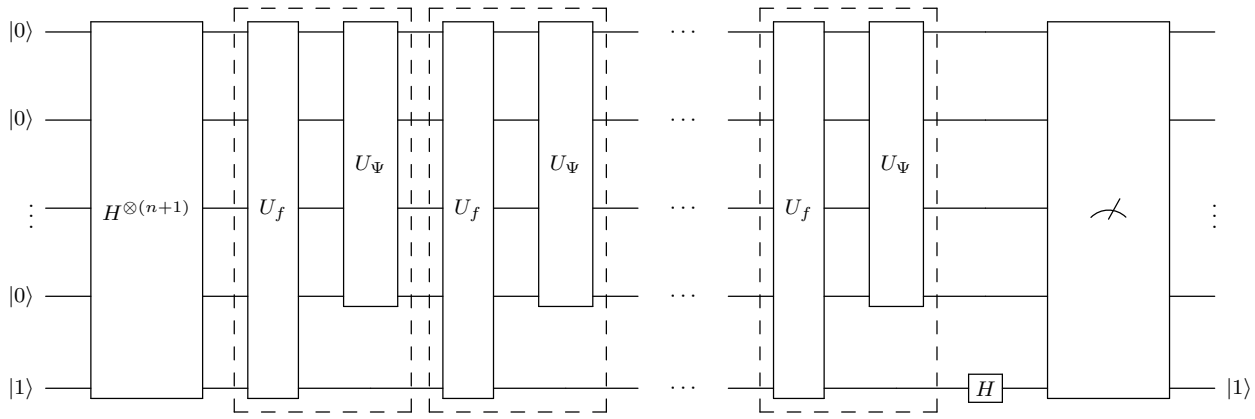


Figure 1: Quantum circuit for Grover's algorithm

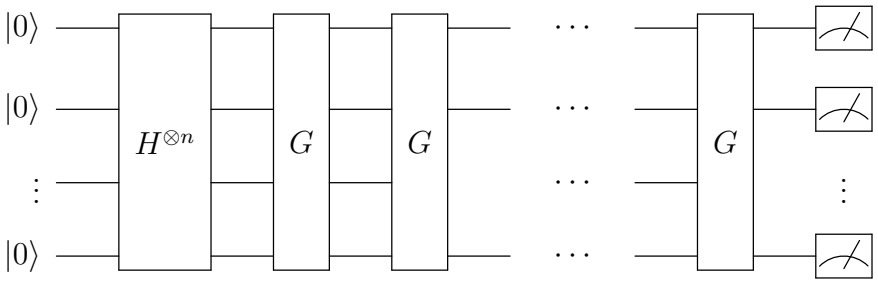


Figure 2: Compact quantum circuit for Grover's algorithm

We already saw that an application of the oracle gate will shift the phase of the searched state. For any given superposition state of n qubits, we would like to know how the effect of the Grover diffusion operator is a reflection about the mean value of the probability amplitudes. Let's consider such a state and adopt the convention of writing the computational basis elements as their integer equivalents, e.g. for $n = 3$, we have $|101\rangle \equiv |5\rangle$.

$$|\Phi\rangle = \sum_{i=0}^{N-1} \phi_i \, |i\rangle$$

It turns out we have the following relation we can use for simplicity:

$$U_\Psi = H^{\otimes n} U_0 H^{\otimes n} \iff U_\Psi = 2 \, |\Psi\rangle\langle\Psi| - I$$

Applying the transformation on the previous state yields

$$U_\Psi \, |\Phi\rangle = 2 \, |\Psi\rangle \, \langle\Psi|\Phi\rangle - |\Phi\rangle \, .$$

The inner product expands to

$$\langle\Psi|\Phi\rangle = \left(\frac{1}{\sqrt{N}} \sum_i \langle i| \right) \left(\sum_i \phi_i \, |i\rangle \right) = \frac{1}{\sqrt{N}} \sum_i \phi_i \, \langle i|i\rangle = \frac{1}{\sqrt{N}} \sum_i \phi_i .$$

The multiplication of $|\Psi\rangle$ by twice this value results in

$$2 \, \langle\Psi|\Phi\rangle \, |\Psi\rangle = \frac{2}{N} \sum_i \phi_i \sum_i |i\rangle = 2\mu \sum_i |i\rangle \, .$$

Finally, we arrive at

$$U_\Psi |\Phi\rangle = 2\mu \sum_i |i\rangle - \sum_i \phi_i |i\rangle = \sum_i (2\mu - \phi_i) |i\rangle = \sum_i (\mu + \mu - \phi_i) |i\rangle .$$

Therefore, the transformation will adjust each probability amplitude by an amount equal to the difference between the mean and itself. When the probability amplitude associated with the solution is first made negative by U_f, the mean will decrease, thus making the difference positive only in this case and negative for all other amplitudes.

Since we do not possess any information about the structure of the database that can help us restrict the search, we can only prepare a uniform superposition as input to the algorithm. During the execution, the element we are looking for will have its probability amplitude increased. Having introduced the necessary concepts, we can now analyze the circuit.

2.1.1 An inductive approach

In the initial superposition state, the coefficient for the solution is labeled ψ_{0*}, while the coefficient of all the other states is ψ_0.

$$|\Psi_0\rangle = \frac{1}{\sqrt{N}} \sum_{\mathbf{x}} |\mathbf{x}\rangle \quad \Longrightarrow \quad U_f |\Psi_0\rangle = - \underbrace{\frac{1}{\sqrt{N}}}_{\psi_{0*}} |\mathbf{x}^*\rangle + (N-1) \underbrace{\frac{1}{\sqrt{N}}}_{\psi_0} \sum_{\mathbf{x} \neq \mathbf{x}^*} |\mathbf{x}\rangle$$

After passing through U_f, the sign of ψ_{0*} is inverted and the mean decreases from the initial $\mu_0 = 1/\sqrt{N}$ to

$$\mu_1 = \frac{1}{N}\Big[(N-1)\psi_0 - \psi_{0*}\Big].$$

The diffusion operator then reflects the coefficients about the new mean value:

$$\psi_{1*} = 2\mu_1 - (-\psi_{0*}) = \frac{1}{\sqrt{N}} \left(\frac{2N-4}{N} + 1 \right) \approx \frac{3}{\sqrt{N}} = 3\,\psi_{0*},$$

$$\psi_1 = 2\mu_1 - \psi_0 = \frac{1}{\sqrt{N}} \left(\frac{2N-4}{N} - 1 \right) \approx \frac{1}{\sqrt{N}} = \psi_0,$$

approximations that hold when N is large enough. We notice that after the first iteration, the probability amplitude of the solution is almost three times higher, while the remaining amplitudes decrease. Following the same procedure,

$$\mu_2 = \frac{1}{\sqrt{N}} \left(\frac{N^2 - 8N + 5}{N} \right) \Longrightarrow \left| \begin{array}{l} \psi_{2*} = 2\mu_2 - (-\psi_{1*}) = \dfrac{1}{\sqrt{N}} \left(\dfrac{5N^2 - 20N + 10}{N^2} \right) \approx 5\,\psi_{0*}, \\ \\ \psi_2 = 2\mu_2 - \psi_1 = \dfrac{1}{\sqrt{N}} \left(\dfrac{N^2 - 12N + 10}{N^2} \right) \end{array} \right.$$

From this pattern, we observe that

$$\psi_{k*} \approx (2k+1)\psi_{0*} = \frac{2k+1}{\sqrt{N}},$$

although it clearly cannot increase boundlessly, as the norm of the state vector is preserved throughout the circuit. In fact, since after each iteration the gain corresponding to the solution surpasses how much the other states are diminished, the mean will continuously decrease, which in turn affects the next iteration. In this manner, when the mean becomes negative, the

algorithm performs worse and it is moving away from finding the solution. Thus, the execution should stop when the mean is very close to zero, which indicates a maximum value for the probability of getting the solution after measurement.

$$|\mu_k| \approx 0 \implies \psi_{k*} \approx 1 \implies |\psi_{k*}|^2 \equiv p_{k*} \approx 1$$

By disregarding the gradual decrease of the mean and we consider the above relation between the amplitude after iteration k and the initial amplitude, we directly obtain

$$\psi_{k*} \approx \frac{2k+1}{\sqrt{N}} = 1 \implies k \approx \frac{\sqrt{N}-1}{2} \approx \frac{\sqrt{N}}{2},$$

which is a good number once rounded off, but it does not get the probability sufficiently close to unity, because the increase in amplitude slows down, making the actual number of required iterations higher.

2.1.2 A more accurate approach

Superposition $|\Psi_0\rangle$ can be expressed in terms of the solution state $|\Psi_s\rangle$ and a superposition of the remaining, non-solution states, $|\Psi_r\rangle$. Considering the normalization constraint the state obeys, the probability amplitudes can be redefined [10].

$$|\Psi_0\rangle = \frac{1}{\sqrt{N}}|\Psi_s\rangle + \sqrt{\frac{N-1}{N}}|\Psi_r\rangle = \sin\theta\,|\Psi_s\rangle + \cos\theta\,|\Psi_r\rangle\,, \quad \theta = \arcsin\frac{1}{\sqrt{N}}$$

After the first Grover iteration is applied and some trigonometric identities are exploited,

$$|\Psi_1\rangle = G\,|\Psi_0\rangle = \Big(2\,|\Psi_0\rangle\langle\Psi_0| - I\Big)\Big(-\sin\theta\,|\Psi_s\rangle + \cos\theta\,|\Psi_r\rangle\Big) = \sin 3\theta\,|\Psi_s\rangle + \cos 3\theta\,|\Psi_r\rangle\,.$$

The state after k Grover iterations then becomes

$$|\Psi_k\rangle = G^k\,|\Psi_0\rangle = \sin\big((2k+1)\theta\big)\,|\Psi_s\rangle + \cos\big((2k+1)\theta\big)\,|\Psi_r\rangle\,.$$

In order to make sure that the measurement is very likely to return $|\Psi_s\rangle = |\mathbf{x}^*\rangle$, it follows that

$$\sin\big((2k+1)\theta\big) \approx 1 \iff (2k+1)\theta \approx \frac{\pi}{2} \implies k \approx \frac{\pi}{4\theta} - \frac{1}{2} \approx \frac{\pi}{4}\sqrt{N},$$

making the ideal number of Grover iterations

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

2.2 Examples and analysis

We now present an implementation of Grover's algorithm using IBM's Qiskit framework [11]. This example considers a variable n-qubit register, with source code given in Listing 1. Since the algorithm itself relies on repeated applications of the same transformation, the implementation takes into account this modular feature and is contains subcircuits, which are then attached together to build the larger circuit.

The program allows us to specify the solution state explicitly or it can be chosen randomly. Either way, the function that defines the effect of the oracle has a general implementation that will properly mark the selected state. Furthermore, the oracle and the diffusion transformation are subcircuits that adapt with the chosen number of qubits n. The algorithm can be either simulated using Qiskit's internal simulator or it can be executed remotely on IBM's available backends, including real devices. Both simulation and real execution results are discussed. The output of the program gives information about the backend that is being used, the searched state, the ideal number of iterations and the associative array corresponding to measurement.

Note: Qiskit's convention associates the last qubit of a register to the topmost qubit of the circuit.

2.2.1 Simulation

The most simple case when $n = 2$ actually indicates an impressive start. After just one iteration, the probability of the solution state is boosted to exactly 1, as Figure 4a depicts below. In this case, state $|10\rangle$ was selected at random and out of all 2000 preparations of the qubit register and runs of the algorithm, the desired state came out every time after measurement. For $n = 3$, the calculated probability of finding the solution after $k = 2$ iterations is about 0.945. One experiment reveals that state $|010\rangle$ is found with probability 0.946, while the others are negligible (Figure 4b).

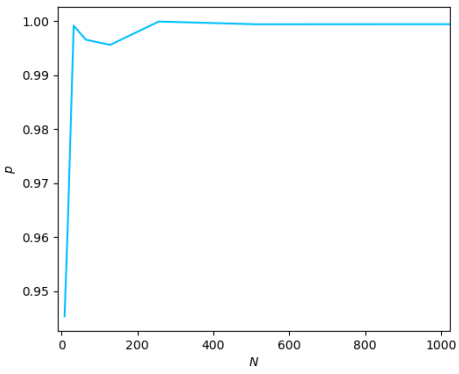
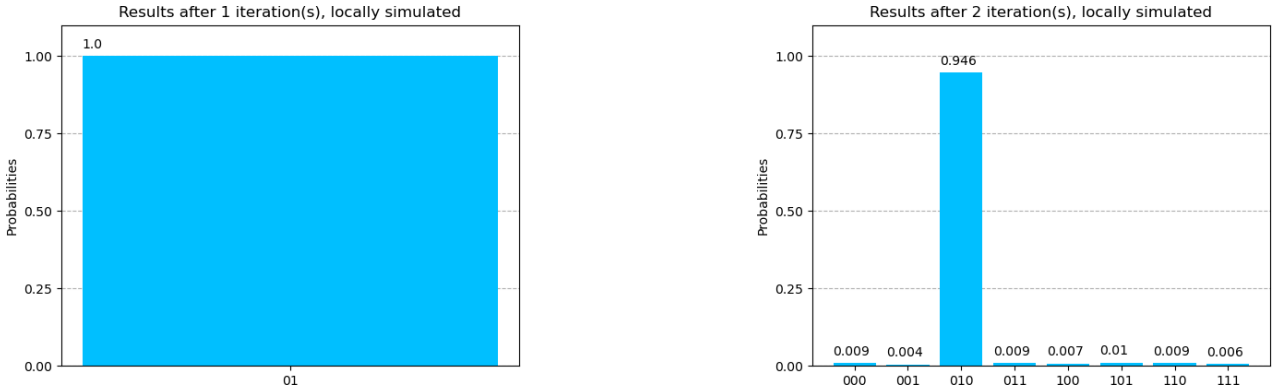


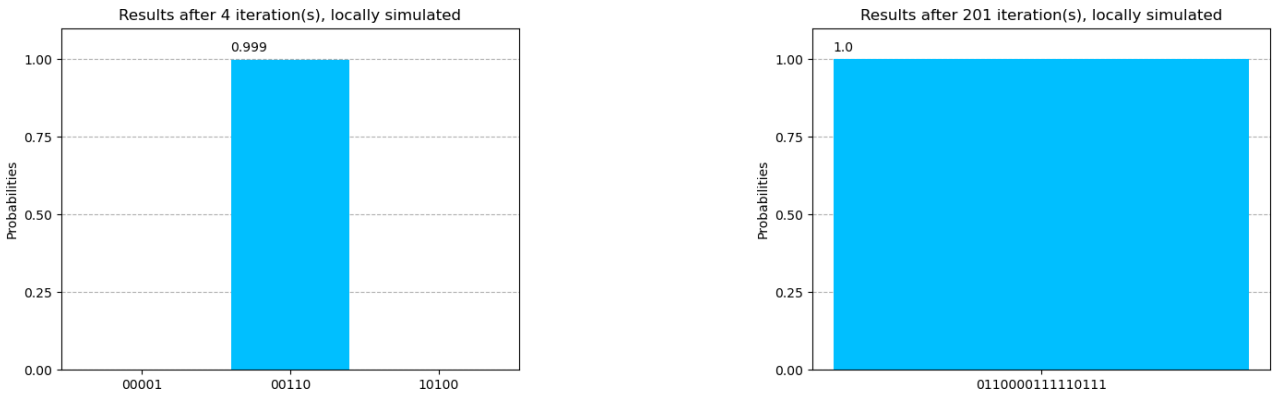
Figure 3: p vs. N graph

As n increases, the probability p associated with the marked state approaches 1, for the same number of iterations k, as displayed by the graph in Figure 3. This is apparent when running the simulation for larger values, for example $n = 5$. The histogram shows that the measurement returned $|01100\rangle$ almost every time (1998 instances), with only two other states observed very rarely (one instance each). For an even larger $n = 16$, some executions can actually return the marked state with probability 1, which justifies the efficiency of the algorithm. Both cases are displayed in Figure 5.



(a) (b)

Figure 4: Simulation of Grover's algorithm for $n = 2$ and $n = 3$

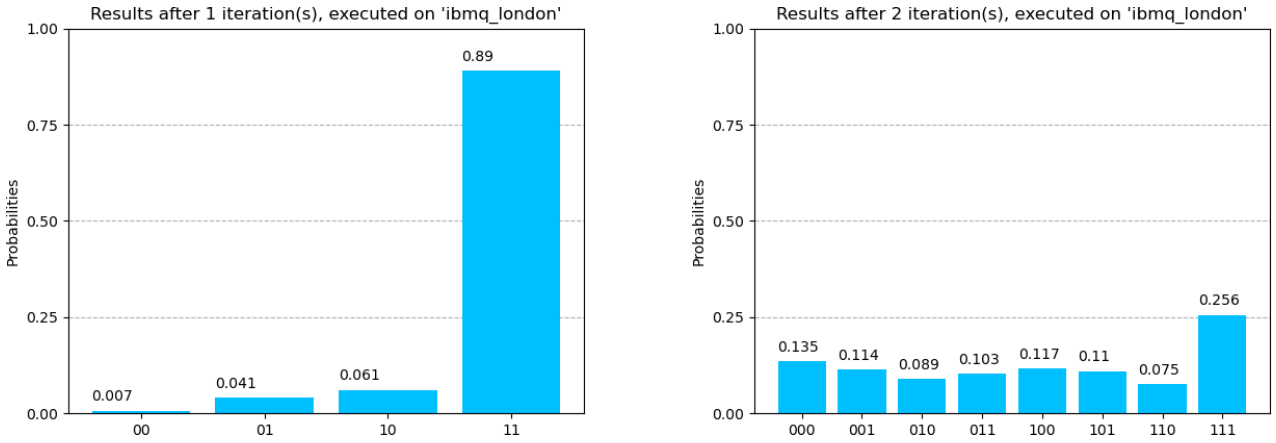


(a) (b)

Figure 5: Simulation of Grover's algorithm for $n = 5$ and $n = 16$

2.2.2 Execution on real devices

Running Grover's algorithm on a real quantum computer comes with reasonable limitations. First of all, the size of the quantum register can be no greater than the actual number of qubits of the physical computer. Besides this, the experiments are no longer noise-free and the performance of the algorithm strongly depends on other factors, such as the partial connectivity of the architecture, qubit quality and gate errors. Each backend has a queue where jobs are pending for execution. In order to manage this aspect more easily, IBM's framework can select the least busy backend when someone decides to run a program. For $n = 2$, this time the algorithm ran on `ibmq_london`, a 5-qubit processor. The results in Figure 6 are visibly different from the ideal case we previously discussed. Now, marked state $|11\rangle$ returns from the 2000 measurements only 1781 times, thus having a probability of about 0.89 instead of exactly 1. The other states have non-zero amplitudes because of the processor's intrinsic computation errors. For $n = 3$, the probability of the state is increased to merely 0.256 after the iterations are finished. The distribution is still unimodal, although the performance degrades once n increases and the solution is no longer distinguishable.



(a) (b)

Figure 6: Real execution of Grover's algorithm for $n = 2$ and $n = 3$

2.3 Qiskit implementation

Listing 1: Grover's algorithm

```
# GROVER'S ALGORITHM

from qiskit import *
from qiskit.compiler import transpile, assemble
from qiskit.providers.ibmq import least_busy
from qiskit.tools.monitor import job_monitor
from qiskit.visualization import plot_histogram

from math import *
from random import randrange
import matplotlib.pyplot as plt
import numpy as np

#================================================================
#=== FUNCTION DEFINITIONS #======================================


```

```
# n-bit binary representation of integer
def bst(n,s):
    return str(bin(s)[2:].rjust(n,'0'))


# subcircuit applying gates given as arguments to every qubit
def gn(n,*args):
    qc = QuantumCircuit(n,n)
    for i in range(n):
        for gate in args:
            getattr(qc, gate)(i)
    return qc

# subcircuit implementing the oracle
def oracle(n,s):
    # adds phase shift only to state corresponding to "s"
    qc = QuantumCircuit(n,n)
    for i in range(n):
        if s[n-1-i] == "0":
            qc.x(i)

    # applies CZ gate with controls 0:n-2 and target n-1
    qc.h(n-1)
    qc.mcx(list(range(n-1)),n-1)
    qc.h(n-1)

    for i in range(n):
        if s[n-1-i] == "0":
            qc.x(i)

    qc.barrier()
    return qc


# subcircuit implementing the diffusion transformation
def diffusion(n):
    # amplifies the probability amplitude of the solution state
    qc = QuantumCircuit(n,n)
    qc += gn(n,"h","x")

    qc.h(n-1)
    qc.mcx(list(range(n-1)),n-1)
    qc.h(n-1)

    qc += gn(n,"x","h")

    qc.barrier()
    return qc

# subcircuit implementing a single Grover iteration
def grover_iteration(n,s):
    qc = QuantumCircuit(n,n)
    qc += oracle(n,s)
    qc += diffusion(n)
    return qc

# custom histogram plot
def show_results(counts,shots,iterations,comment):
    states = list(counts.keys())
    outcomes = list(counts.values())
```

```
    prob = [round(i/shots,3) for i in outcomes]

    d = dict(zip(states,prob))
    d = dict(sorted(d.items()))
    states = list(d.keys())
    prob = list(d.values())

    bp = plt.bar(states,prob,color='deepskyblue',zorder=2)

    for i in bp:
      h = i.get_height()
      if h > 0.001:
        plt.text(i.get_x(), h + 0.025, h)

    plt.ylim(0,1.1)
    plt.yticks(np.arange(0,1.25,0.25))
    plt.ylabel("Probabilities")
    plt.grid(axis='y',linestyle='dashed',zorder=0)
    plt.title("Results after " + str(iterations) + " iteration(s), " + comment)
    plt.show()

#================================================================
#=== INITIAL PARAMETERS #=======================================

# size of quantum and classical registers
n = 3

# size of the search space
N = 2**n

# ideal number of iterations
k = floor(pi/4*sqrt(N))

#=== EXPERIMENT TYPE #==========================================

# local simulation, remote simulation, execution on real device
exp_type = "real"

if exp_type == "local":
    backend = Aer.get_backend('qasm_simulator')

else:
    provider = IBMQ.load_account()
    if exp_type == "real":
        backend = least_busy(provider.backends(filters=lambda x:
        ↪  x.configuration().n_qubits >= n and
                                            not x.configuration().simulator and
                                            ↪  x.status().operational==True))
    elif exp_type == "hpc":
        backend = provider.get_backend('ibmq_qasm_simulator')

#================================================================
#=== SEARCHED ELEMENT #=========================================
s = randrange(N)
s = bst(n,s)
#s = "001"
print("BACKEND: " + str(backend))
print("SEARCHING FOR |" + s + ">")
print("REQUIRED ITERATIONS: " + str(k))

```

```
#================================================================

grc = QuantumCircuit(n,n)

# CREATE INITIAL SUPERPOSITION
grc += gn(n,"h")

# REPEAT UNTIL THE IDEAL NUMBER OF ITERATIONS IS REACHED
for iteration in range(k):
    grc += grover_iteration(n,s)

# MEASURE AT THE END
grc.measure(list(range(n)),list(range(n)))

# number of circuit instances to be measured
shots = 2000

if exp_type == "local":
    result = execute(grc,backend,shots=shots).result()
    comment = "locally simulated"

elif exp_type == "real" or exp_type == "hpc":
    job_grc = execute(grc, backend, shots=shots)
    job_monitor(job_grc)
    result = job_grc.result()
    comment = "executed on '" + str(backend) + "'"

counts = result.get_counts(grc)
print("RESULTS: " + str(counts))

show_results(counts,shots,k,comment)
#plot_histogram(counts, title="Results after " + str(k) + " iteration(s), " + comment)
```

3 Conclusions

The robustness of the existing information security framework has been challenged ever since the theoretical foundations of quantum computation were laid. With the promising advancements in the realization of quantum computation devices, the threat to modern cryptographic schemes becomes an issue of importance for the future. Even though Grover's algorithm cannot be fully exploited yet because of the practical limitations of the underlying device that runs it, its theoretical efficiency makes it a highly valuable tool as a searching algorithm. From a security perspective, it is even more captivating, as it significantly improves the primary and most straightforward type of attack against symmetric ciphers.

References

[1] Whitfield Diffie and Martin Hellman. "New directions in cryptography". In: *IEEE transactions on Information Theory* 22.6 (1976), pp. 644–654.

[2] Ralph C Merkle. "Secure communications over insecure channels". In: *Communications of the ACM* 21.4 (1978), pp. 294–299.

[3] Ronald L Rivest, Adi Shamir, and Leonard Adleman. "A method for obtaining digital signatures and public-key cryptosystems". In: *Communications of the ACM* 21.2 (1978), pp. 120–126.

[4] Joan Daemen and Vincent Rijmen. "The Block Cipher Rijndael". In: *Smart Card Research and Applications*. Ed. by Jean-Jacques Quisquater and Bruce Schneier. Berlin, Heidelberg: Springer Berlin Heidelberg, 2000, pp. 277–284. ISBN: 978-3-540-44534-0.

[5] Peter W Shor. "Algorithms for quantum computation: discrete logarithms and factoring". In: *Proceedings 35th annual symposium on foundations of computer science*. Ieee. 1994, pp. 124–134.

[6] Lov K Grover. "A fast quantum mechanical algorithm for database search". In: *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*. 1996, pp. 212–219.

[7] Daniel J Bernstein. "Grover vs. McEliece". In: *International Workshop on Post-Quantum Cryptography*. Springer. 2010, pp. 73–80.

[8] Daniel J Bernstein and Tanja Lange. "Post-quantum cryptography". In: *Nature* 549.7671 (2017), pp. 188–194.

[9] Diana Maimuţ and Emil Simion. "Post-quantum Cryptography and a (Qu)Bit More". In: *International Conference on Security for Information Technology and Communications*. Springer. 2018, pp. 22–28.

[10] Phillip Kaye, Raymond Laflamme, Michele Mosca, et al. *An introduction to quantum computing*. Oxford university press, 2007.

[11] Héctor Abraham et al. *Qiskit: An Open-source Framework for Quantum Computing*. 2019. DOI: 10.5281/zenodo.2562110.