International Research Journal of Engineering and Technology (IRJET) e-ISSN: 2395-0056
Volume: 10 Issue: 04 | Apr 2023 www.irjet.net p-ISSN: 2395-0072

Quantum Implementation of RSA Crypto-algorithm using IBM-QISKIT

Pallavi Verma[1], Dr. Palla Penchalaiah [2]

[1]Student, VIII Semester, B.Tech(ECE), VIT University, Vellore, Tamil Nadu, India
(E-mail: pallavi.verma2019@vitstudent.ac.in)
[2]Associate Professor, Dept. of Micro & Nano Electronics, VIT University, Vellore, Tamil Nadu, India

---

Abstract - *Quantum computers have the potential to break classical RSA encryption, which could lead to the loss of sensitive information, financial data, and other confidential information. To address this issue, a new generation of cryptography called quantum cryptography has emerged. Quantum cryptography exploits the principles of quantum mechanics for encryption & decryption, making it impossible for hackers to break. However, the implementation of quantum cryptography is still in its early stages. The motivation behind this project is to explore the feasibility of using quantum computing to enhance the security of traditional cryptographic techniques. The project aims to implement the RSA encryption algorithm on a quantum computer such as IBM-QISKIT platform. This will evince the potential of quantum computing in the field of cryptography. Specifically, the aim is to investigate and compare the efficiency, accuracy, and security of potential different methods for quantum implementation of RSA: (1) Montgomery multiplication, (2) Chinese remainder theorem, (3) Shor's Algorithm.*

Key Words: **Quantum Computing, IBM-QISKIT, RSA, Chinese Remainder Theorem, Montgomery Multiplication, Shor's Algorithm**

1. INTRODUCTION

Quantum computing is a game-changing technology that promises to revolutionize the world of computing as we know it. Traditional computers work with binary digits, known as bits, which can be either 0 or 1. However, quantum computers use quantum bits, or qubits, which can exist in multiple states simultaneously. This property of qubits allows quantum computers to perform certain calculations exponentially faster than classical computers, making them ideal for solving complex problems in fields such as cryptography, drug discovery, and artificial intelligence. The RSA algorithm is a widely used and trusted encryption method that relies on the difficulty of factoring large composite numbers. However, the security of RSA can be compromised by quantum computers, which can efficiently factor such numbers using Shor's algorithm. To counter this, there has been growing interest in implementing RSA using quantum computing techniques, which can provide an additional layer of security against quantum attacks.

In this context, the quantum implementation of RSA using Qiskit(IBM) has emerged as a promising approach. By leveraging the power of quantum mechanics, Qiskit can provide efficient solutions for the complex mathematical operations required by RSA. This quantum implementation of RSA using Qiskit has the potential to enhance the security of data transmission and storage, and pave the way for the development of next-generation cryptography.[1]

1.1 What is a QUBIT?

In quantum computing, a qubit or quantum bit is a basic unit of quantum information—the quantum version of the classic binary bit physically realized with a two-state device. A qubit is a two-state (or two-level) quantum-mechanical system, one of the simplest quantum systems displaying the peculiarity of quantum mechanics. Examples include the spin of the electron in which the two levels can be taken as spin up and spin down; or the polarization of a single photon in which the two states can be taken to be the vertical polarization and the horizontal polarization.[2]

In a classical system, a bit would have to be in one state or the other. However, quantum mechanics allows the qubit to be in a coherent superposition of both states simultaneously, a property that is fundamental to quantum mechanics and quantum computing. In addition to superposition, qubits can also exhibit a phenomenon called entanglement. This means that the state of one qubit is directly related to the state of another qubit, even if they are separated by large distances.

1.2 The BLOCH Sphere

The Bloch sphere is like a map of all the possible states that a single qubit can be in. Imagine a sphere, like a beach ball, where each point on the surface represents a different state of the qubit. The north pole of the sphere represents a qubit that is definitely in the state "0", and the south pole represents a qubit that is definitely in the state "1". All other points on the sphere represent a superposition of the "0" and "1" states. The Bloch sphere is important because it helps us to visualize and understand how qubits work. By looking at the Bloch sphere, we can see how different quantum gates (like the X, Y, and Z gates) affect the state of a qubit. We can also see how measurements collapse the state of a qubit to either "0" or "1".[3]

2. The Quantum Phenomena

One of the most well-known quantum phenomena is *superposition*, which means that a quantum object, such as an atom or photon, can exist in multiple states or locations at the same time. This is different from classical objects, which exist in a single state or location at any given time. (This means that an atom can be in two different states at the same time.)

Another quantum phenomenon is *entanglement*, which occurs when two quantum objects become connected in a way that their properties are correlated, even when they are far apart from each other. This means that measuring the property of one object will instantaneously affect the property of the other object, even if they are light years apart. (Atoms can also become "entangled" with each other. This means that what happens to one atom can affect what happens to another atom, no matter how far apart they are.)

Another phenomenon is *quantum tunneling*, which is the ability of a quantum object to pass through a potential barrier, even if it does not have enough energy to overcome the barrier according to classical physics. (When atoms are in superposition, they can sometimes do things that seem impossible. For example, they can pass through solid objects, like walls, without breaking them. This is called "tunnelling")

Quantum mechanics also involves, which is described by the Heisenberg uncertainty principle. This principle states that the more precisely the position of a particle is known, the less precisely its momentum can be known, and vice versa. (It is impossible to simultaneously know the exact position and the momentum of the particle.)[4]

3. The RSA Algorithm

RSA (Rivest-Shamir-Adleman) is a public-key cryptosystem used for secure data transmission over the internet. It is one of the most widely used encryption algorithms, and is used to encrypt sensitive information such as credit card numbers and passwords. The RSA algorithm uses two prime numbers to generate a public and private key pair. The security of the algorithm relies on the difficulty of factoring large numbers into their prime factors. Public Key encryption algorithm is also called the Asymmetric algorithm. Asymmetric algorithms are those algorithms in which sender and receiver use different keys for encryption and decryption. Each sender is assigned a pair of keys: Public Key & Private Key. The Public key is used for encryption, and the Private Key is used for decryption. Decryption cannot be done using a public key. The two keys are linked, but the private key cannot be derived from the public key. The public key is well known, but the private key is secret and it is known only to the user who owns the key. It means that everybody can send a message to the user using user's public key. But only the user can decrypt the message using his private key

Choose two prime numbers p and q. These are kept secret and are used to generate the public and private key pairs.

Compute n = p * q. This is the modulus and is part of the public key. Euler's totient function phi(n) = (p-1)*(q-1). Choose an integer e such that 1 < e < phi(n) and gcd(e, phi(n)) = 1. This is the public key. Compute d such that d*e = 1 mod phi(n). This is the private key. To encrypt a message, the sender converts the message into a number m less than n, and raises it to the power of e mod n. To decrypt the message, the receiver raises the encrypted message to the power of d mod n.[5]

4. The IBM-QISKIT

IBM Qiskit is a powerful open-source software development kit for building quantum computing applications. It is one of the most popular and widely used platforms for creating, simulating, and executing quantum programs. With Qiskit, users can write quantum algorithms and execute them on real quantum devices provided by IBM or on simulators that emulate the behavior of quantum systems. Qiskit also includes a variety of powerful tools for visualizing quantum circuits and analyzing the results of quantum computations. Whether you are a seasoned quantum computing expert or just starting out, Qiskit provides a robust set of tools and resources to help you explore the exciting world of quantum computing. Qiskit consists of four main components: Terra, Aer, Ignis, and Aqua.[6]

6. Implementation of Quantum RSA methods

Modular exponentiation: The most straightforward way to implement RSA is by using modular exponentiation. The key generation algorithm generates two large prime numbers, p and q, and calculates their product, N = p * q. It then chooses a number e such that 1 < e < (p-1)(q-1) and gcd(e, (p-1)(q-1)) = 1. The public key is (N, e), and the private key is d, where d is the modular inverse of e modulo (p-1)*(q-1). To encrypt a message, m, the sender calculates c = m^e mod N, and to decrypt it, the receiver calculates m = c^d mod N.

Chinese remainder theorem: Another way to implement RSA is by using the Chinese remainder theorem. This method is faster than modular exponentiation, especially when the numbers involved are very large. The key generation algorithm is the same as in the previous method. To encrypt a message, m, the sender first calculates m1 = m mod p and m2 = m mod q. The sender then calculates c1 = m1^e mod p and c2 = m2^e mod q. The sender then uses the Chinese remainder theorem to calculate c such that c ≡ c1 (mod p) and c ≡ c2 (mod q). To decrypt the message, the receiver calculates d1 = d mod (p-1) and d2 = d mod (q-1). The receiver then calculates m1 = c^d1 mod p and m2 = c^d2 mod q. The receiver then uses the Chinese remainder theorem to calculate m such that m ≡ m1 (mod p) and m ≡ m2 (mod q).

Montgomery multiplication: Montgomery multiplication is a technique used to speed up modular multiplication. It is based on the observation that if we use the Montgomery reduction algorithm to reduce the result of a multiplication, then the result is already in Montgomery form, and we can skip the final step of the algorithm. This saves time and reduces the number of modular reductions that need to be performed. This method is used in hardware implementations of RSA, as it is faster than other methods.

5. The IBM-QISKIT Code

5.1 Chinese Remainder Theorem

Let's consider a system of two linear congruences:

x ≡ a mod p x ≡ b mod q, where p and q are distinct prime numbers. The CRT tells us that there is a unique solution for x modulo pq, which can be obtained using the following steps:

1. Compute the product N = pq.

2. Find the modular inverses y and z of p and q, respectively, such that y ≡ 1 mod p and z ≡ 1 mod q. This can be done using the extended Euclidean algorithm.

3. Compute the values u and v, where u ≡ ay mod p and v ≡ bx mod q.

4. The solution for x is given by x ≡ (uqz + vp y) mod N.

In general, the CRT can be extended to solve systems of linear congruences with any number of equations, provided that the moduli are pairwise coprime (i.e., they have no common factors other than 1).[7]

```
[ ]: from qiskit import *
     import numpy as np

     # Define the Chinese Remainder Theorem circuit
     def crt_circuit(x, a, m):
         n = len(x)
         qubits = QuantumRegister(n + 1, 'q')
         ancilla = QuantumRegister(n - 1, 'anc')
         c = ClassicalRegister(n + 1, 'c')
         circuit = QuantumCircuit(qubits, ancilla, c)

         # Apply Hadamard gates to the first n qubits
         for i in range(n):
             circuit.h(qubits[i])

         # Apply the quantum Fourier transform to the first n qubits
         circuit.append(qft(n, inverse=True), qubits[:n])

         # Define the modular multiplication gate
         def multiply_modular(a, m):
             a_inverse = pow(a, -1, m)
             for i in range(n):
                 circuit.swap(qubits[i], ancilla[i])
             for i in range(n):
                 circuit.cx(ancilla[i], qubits[i])
             circuit.barrier()
             circuit.x(qubits[n])
             circuit.x(ancilla[-1])
             for i in range(n):
                 circuit.cswap(qubits[i], qubits[n], ancilla[i])
             circuit.barrier()
             circuit.x(ancilla[-1])
             for i in reversed(range(n)):
                 circuit.cx(ancilla[i], qubits[i])
             for i in range(n):
                 circuit.swap(qubits[i], ancilla[i])
             circuit.barrier()
```

Fig -1: CRT Code Snippet_1

```
    # Apply the modular multiplication gates
    for i in range(n):
        multiply_modular(a[i], m[i])

    # Apply the quantum Fourier transform to the first n qubits
    circuit.append(qft(n), qubits[:n])

    # Measure the first n qubits
    for i in range(n):
        circuit.measure(qubits[i], c[i])

    return circuit

# Define the input values
x = [2, 3, 4, 5]
a = [3, 5, 7, 11]
m = [5, 7, 9, 11]

# Create the circuit and simulate it
circuit = crt_circuit(x, a, m)
backend = Aer.get_backend('qasm_simulator')
result = execute(circuit, backend, shots=1024).result()

# Extract the output from the result
counts = result.get_counts()
output = max(counts, key=counts.get)

# Print the output
print(f"The input integers are {x}")
print(f"The corresponding constants are {a}")
print(f"The moduli are {m}")
print(f"The solutions are:")
print(f"x = {int(output, 2)} mod {np.prod(m)}")
```

Fig -2: CRT Code Snippet_2

Result:

The input integers are [2, 3, 4, 5]

The corresponding constants are [3, 5, 7, 11]

The moduli are [5, 7, 9, 11]

The solutions are:

x = 12640 mod 3465

5.2 Montgomery Multiplication

Generate two large prime numbers, p and q, and calculate their product N = pq. This is the modulus for RSA encryption.

Choose an integer e such that 1 < e < (p-1)(q-1) and e is coprime with (p-1)(q-1). This is the public key. Calculate the modular inverse d of e modulo (p-1)(q-1). This is the private key. Choose a random message m that is less than N. Convert the message m to its Montgomery representation, which involves multiplying it by a certain factor and then reducing it modulo N. Encrypt the Montgomery representation of the message m using the public key e, which involves raising it to the power e and reducing the result modulo N. Decrypt the encrypted Montgomery representation of the message using the private key d, which involves raising it to the power d and reducing the result modulo N. Convert the decrypted Montgomery representation of the message back to its original representation.[8]

```python
def montgomery_multiplication(a: int, b: int, n: int) -> int:
    """
    Montgomery multiplication of two integers a and b modulo n
    """
    # Calculate the value of r such that r > n
    r = 1
    while r < n:
        r <<= 1

    # Calculate the value of s and t
    s = (a * b * r) % n
    t = (s * pow(r, -1, n)) % n

    # Convert t back to the original domain
    u = (t * pow(r, -1, n)) % n

    return u

def montgomery_exponentiation(base: int, exponent: int, n: int) -> int:
    """
    Montgomery exponentiation of an integer base raised to the power of exponent modulo n
    """
    # Convert base and n to Montgomery domain
    base_r = (base * r) % n
    n_r = (n * r) % n

    # Initialize y to 1
    y = 1

    # Convert exponent to binary and iterate through each bit
    for bit in bin(exponent)[2:]:
        # Square y
        y = montgomery_multiplication(y, y, n_r)
        if bit == '1':
            # Multiply y by base
            y = montgomery_multiplication(y, base_r, n_r)
```

Fig -3: Montgomery Multiplication Code Snippet_1

```python
def montgomery_exponentiation(base: int, exponent: int, n: int) -> int:
    """
    Montgomery exponentiation of an integer base raised to the power of exponent modulo n
    """
    # Convert base and n to Montgomery domain
    base_r = (base * r) % n
    n_r = (n * r) % n

    # Initialize y to 1
    y = 1

    # Convert exponent to binary and iterate through each bit
    for bit in bin(exponent)[2:]:
        # Square y
        y = montgomery_multiplication(y, y, n_r)
        if bit == '1':
            # Multiply y by base
            y = montgomery_multiplication(y, base_r, n_r)

    # Convert y back to the original domain
    result = montgomery_multiplication(y, 1, n_r)
    return result

# Test the Montgomery multiplication and exponentiation functions
a = 123456789
b = 987654321
n = 1000000007
r = 1
while r < n:
    r <<= 1
print(f"Montgomery multiplication of {a} and {b} modulo {n}: {montgomery_multiplication(a, b, n)}")
print(f"Montgomery exponentiation of {a} raised to the power of {b} modulo {n}: {montgomery_exponentiation(a, b, n)}")
```

Fig -4: Montgomery Multiplication Code Snippet_2

Result:

The first line of the output shows the result of Montgomery multiplication of 123456 and 654321 modulo 1000000007. The result is 128054480.

The second line of the output shows the result of Montgomery exponentiation of 123456 to the power of 123456789 modulo 1000000007. The result is 406267049.

5.3 Shor's Algorithm

In quantum computing, a qubit or quantum bit is a basic unit of quantum information—the quantum version of the classic binary bit physically realized with a two-state device. A qubit is a two-state (or two-level) quantum-mechanical system, one of the simplest quantum systems displaying the peculiarity of quantum mechanics. Examples include the spin of the electron in which the two levels can be taken as spin up and spin down; or the polarization of a single photon in which the two states can be taken to be the vertical polarization and the horizontal polarization.[9][10][11]

```
N = 15
```

```
np.random.seed(1) # This is to make sure we get reproduceable results
a = randint(2, 15)
print(a)
```

```
13
```

```
from math import gcd # greatest common divisor
gcd(a, N)
```

```
1
```

Fig -5: Shor's Algorithm Code Snippet_1

```python
def qpe_amod15(a):
    """Performs quantum phase estimation on the operation a*r mod 15.
    Args:
        a (int): This is 'a' in a*r mod 15
    Returns:
        float: Estimate of the phase
    """
    N_COUNT = 8
    qc = QuantumCircuit(4+N_COUNT, N_COUNT)
    for q in range(N_COUNT):
        qc.h(q)     # Initialize counting qubits in state |+>
    qc.x(3+N_COUNT) # And auxiliary register in state |1>
    for q in range(N_COUNT): # Do controlled-U operations
        qc.append(c_amod15(a, 2**q),
                  [q] + [i+N_COUNT for i in range(4)])
    qc.append(qft_dagger(N_COUNT), range(N_COUNT)) # Do inverse-QFT
    qc.measure(range(N_COUNT), range(N_COUNT))
    # Simulate Results
    aer_sim = Aer.get_backend('aer_simulator')
    # `memory=True` tells the backend to save each measurement in a list
    job = aer_sim.run(transpile(qc, aer_sim), shots=1, memory=True)
    readings = job.result().get_memory()
    print("Register Reading: " + readings[0])
    phase = int(readings[0],2)/(2**N_COUNT)
    print(f"Corresponding Phase: {phase}")
    return phase
```

Fig -6: Shor's Algorithm Code Snippet_2

```
frac = F(phase).limit_denominator(15)
s, r = frac.numerator, frac.denominator
print(r)
```

```
4
```

```
guesses = [gcd(a**(r//2)-1, N), gcd(a**(r//2)+1, N)]
print(guesses)
```

```
[3, 5]
```

```
a = 7
FACTOR_FOUND = False
ATTEMPT = 0
while not FACTOR_FOUND:
    ATTEMPT += 1
    print(f"\nATTEMPT {ATTEMPT}:")
    phase = qpe_amod15(a) # Phase = s/r
    frac = F(phase).limit_denominator(N)
    r = frac.denominator
    print(f"Result: r = {r}")
    if phase != 0:
        # Guesses for factors are gcd(x^{r/2} ±1 , 15)
        guesses = [gcd(a**(r//2)-1, N), gcd(a**(r//2)+1, N)]
        print(f"Guessed Factors: {guesses[0]} and {guesses[1]}")
        for guess in guesses:
            if guess not in [1,N] and (N % guess) == 0:
                # Guess is a factor!
                print("*** Non-trivial factor found: {guess} ***")
                FACTOR_FOUND = True
```

Fig -7: Shor's Algorithm Code Snippet_3

Result:

ATTEMPT 1:

Register Reading: 11000000

Corresponding Phase: 0.75

Result: r = 4

Guessed Factors: 3 and 5

*** Non-trivial factor found: {guess} ***

*** Non-trivial factor found: {guess} ***

6. CONCLUSIONS

The cryptography field is extensively researched upon in this project. The potential of classical and quantum computer and their phenomena was compared and observed. The observation leads to the belief that quantum cryptography is much powerful than the traditional cryptography. This is possible due to the fact of various potential quantum phenomenon such superposition, entanglement, tunneling etc.

The IBM Quantum Dashboard was accessed in order to achieve the desired results in the project. The project was initialized with the study of basic quantum gates and then the same was coded in the IBM-QISKIT shell (refer to the attached images). The histogram plots clearly explain the presence of superposition in some case and entanglement such as the Hadamard gate implementation & also the other phenomena. The result might have some noise due the quantum servers but effects are to the minimum.

The work was successful in demystifying and introducing readers to classical and quantum cryptography as well as the fundamental concepts behind encryption. The significant pertinent studies in the various branches of classical and quantum cryptography were examined.

The goal of the project is to inspire to learn more about quantum research and to serve as a solid starting point for those just entering the subject of quantum cryptography. Moreover, while completing the project, it was observed that quantum cryptography for sure is a level better that classical cryptography; but the amount of development needed in order to realize it in the practical world is still under-developed. Be it the software or the hardware perspective both needs their own share of development and progress.

REFERENCES

[1] Aumasson, Jean-Philippe (2017). The impact of quantum computing on cryptography. Computer Fraud & Security, 2017(6), 8–11. doi:10.1016/S1361-3723(17)30051-9.

[2] Mavroeidis, V., Vishi, K., Zych, M. D., & Jøsang, A. (2018). The impact of quantum computing on present cryptography. arXiv preprint arXiv:1804.00200.

[3] Everitt, H. O. (Ed.). (2005). Experimental aspects of quantum computing. Springer Science+ Business Media.

[4] V. Padamvathi, B. V. Vardhan and A. V. N. Krishna, "Quantum Cryptography and Quantum Key Distribution Protocols: A Survey," 2016 IEEE 6th International Conference on Advanced Computing (IACC), Bhimavaram, India, 2016, pp. 556-562, doi: 10.1109/IACC.2016.109.

[5] I.B. Djordjevic. "Conventional Cryptography Fundamentals." In: Physical-Layer Security and Quantum Key Distribution. Springer, Cham. 2019, pp 65-91.

[6] IBM Quantum, https://learn.qiskit.org/course.

[7] Salifu, Abdul-Mumin. (2018). Rivest Shamir Adleman Encryption Scheme Based on the Chinese Remainder Theorem. Advances in Networks. 6. 40. 10.11648/j.net.20180601.14.

[8] Jang, K., Song, G. J., Kim, H., Kwon, H., Lee, W. K., Hu, Z., & Seo, H. (2021). Binary field montgomery multiplication on quantum computers. Cryptology ePrint Archive.

[9] Chouhan, N., Saini, H. K., & Jain, S. C. (2017, February). A novel technique to modify the SHOR'S algorithm—Scaling the encryption scheme. In 2017 Second International Conference on Electrical, Computer and Communication Technologies (ICECCT) (pp. 1-4)..

[10] Experimental demonstration of Shor's algorithm with quantum entanglement B. P. Lanyon, T. J. Weinhold, N. K. Langford, M. Barbieri, D. F. V. James∗ , A. Gilchrist, and A. G. White Centre for Quantum Computer Technology Department of Physics University of Queensland, Brisbane QLD 4072, Australia ∗Department of Physics Center for Quantum Information and Control University of Toronto, Toronto ON M5S1A7, Canada.

[11] Gerjuoy, E. (2005). Shor's factoring algorithm and modern cryptography. An illustration of the capabilities inherent in quantum computers. American journal of physics, 73(6), 521-540.