Breaking RSA with Quantum Fourier Transforms

IC121 Applications of Particles and Waves

Group Assignment 1

Group Members

Abhishek Mantri	B23060
Harshit Jain	B23132
Arani Ghosh	B23119
Jigar Fufal	B24244
Arka Mukhopadhay	B23120
Ojas More	B24263
Parth Gawande	B24266

April 23, 2025

Contents

1 What is RSA?

RSA is an asymmetric cryptography algorithm widely used for secure data transmission. Asymmetric, or public-key cryptography, uses two different keys: a public key for encryption and a private key for decryption. The public key can be freely distributed, while the private key must be kept secret.

1.1 RSA Key Generation

Algorithm 1 RSA Key Generation

```
1: Choose two distinct large prime numbers, $p$ and $q$.
2: Compute the modulus $n = p \times q$. This $n$ is part of both the public and private keys.
3: Calculate Euler's totient function: $\varphi(n) = (p-1)(q-1)$.
4: Choose an integer $e$ such that $1 < e < \varphi(n)$ and $\gcd(e, \varphi(n)) = 1$. This $e$ is the public exponent.
5: Compute an integer $d$ such that $(d \times e) \equiv 1 \pmod{\varphi(n)}$. This $d$ is the private exponent.
```

The public key is (n, e). The private key is (n, d). The primes p and q, and $\varphi(n)$ should also be kept secret.

1.2 Encryption and Decryption

Encryption: To encrypt a message m (with $0 \leq m < n$), compute the ciphertext c as:

$$c \equiv m^e \pmod{n}$$

Decryption: To decrypt c, compute:

$$m \equiv c^d \pmod{n}$$

2 Weaknesses of RSA

Despite its widespread use, RSA has several vulnerabilities:

- **Poor Choice of Primes:** If the primes p and q are chosen poorly, factorization becomes easier. For example, if p and q are too close to each other (e.g., $|p - q| \ll \sqrt{n}$), Fermat's factorization method becomes efficient. Similarly, if one prime is significantly smaller than the other, Elliptic Curve Factorization (ECF) can effectively factor the modulus.
- **Small Modulus:** If the modulus n is too small, it can be factored easily. Secure RSA uses moduli of at least 2048 bits.
- **Low Public Exponent:** A small exponent e (e.g., 3) may speed up encryption, but if used improperly (e.g., encrypting the same message with the same e and different moduli), it can allow recovery of the plaintext using techniques like the Chinese Remainder Theorem. Also, if $m^e < n$, the ciphertext $c = m^e$ leaks m directly.
- **Weak Random Number Generation:** Poor or predictable generation of primes p and q can allow attackers to factor n and break the encryption.
- **Side-Channel Attacks:** These exploit physical characteristics like timing or power usage during encryption/decryption to gain insights into the private key.
- **Advanced Factoring Algorithms:** Techniques such as the General Number Field Sieve (GNFS) continue to improve and pose a threat to RSA's long-term security.
- **Quantum Threats:** Algorithms like Shor's, running on quantum computers, can factor large integers efficiently, potentially rendering RSA obsolete.

3 Fermat's Factorization for Close Primes

Definition 1 (Fermat's Factorization)**.** *If $n = a^2 - b^2 = (a-b)(a+b)$, then $p = a - b$ and $q = a + b$ are factors of n.*

Fermat's factorization method exploits a fundamental algebraic identity: the difference of squares. When an RSA modulus $n = pq$ can be expressed as $a^2 - b^2$ for some integers a and b, we can immediately factor n as $(a-b)(a+b)$.

Algorithm 2 Fermat's Factorization Method

```
1: Start with $a = \lceil \sqrt{n} \rceil$
2: Calculate $b^2 = a^2 - n$
3: Check if $b^2$ is a perfect square
4: If yes, compute $p = a - b$ and $q = a + b$
5: If no, increment $a$ and repeat from step 2
```

Exploiting Close Primes: If p and q are close to each other, this method becomes highly efficient. When $p \approx q$, their arithmetic mean $\frac{p+q}{2}$ is approximately $\sqrt{n}$, making $a \approx \sqrt{n}$. The search for a typically requires very few iterations, making the factorization trivial.
This vulnerability highlights why proper RSA implementations should use primes that are sufficiently far apart. Modern standards recommend that $|p - q| > 2^{(n/2-100)}$ where n is the bit length of the modulus.

Example 1. *Suppose $n = 851$. Then $\sqrt{851} \approx 29.17$.*
Try $x = 30$: $30^2 - 851 = 900 - 851 = 49 = 7^2$.
So $x = 30$, $y = 7$, $p = 30 - 7 = 23$, $q = 30 + 7 = 37$.
Indeed, $23 \times 37 = 851$.

```
def fermat_factorization(n):

    # Start with the ceiling of sqrt(n)
    a = ceil(sqrt(n))

    # Initialize b_squared = a^2 - n
    b_squared = a^2 - n

    # Continue until we find a perfect square
    while not is_square(b_squared):
        a += 1
        b_squared = a^2 - n

    # Once we find a perfect square, compute b
    b = sqrt(b_squared)

    # Return the factors
    return (a - b, a + b)

n = 5959
p, q = fermat_factorization(n)
print(f"Factors of {n}: {p} and {q}")
print(f"Verification: {p} * {q} = {p*q}")
```

Listing 1: SageMath Implementation of Fermat's Factorization Theorem

This implementation demonstrates Fermat's factorization method, which is particularly effective when the prime factors are close together. The algorithm starts with the ceiling of the square root of the modulus and increments until it finds a value that makes $a^2 - n$ a perfect square, allowing for easy factorization.

4 Elliptic Curve Factorization (ECF)

Definition 2. *Elliptic Curve Factorization (ECF), developed by Hendrik Lenstra, is a probabilistic algorithm whose running time primarily depends on the size of the smallest prime factor of n, rather than the size of n itself.*

ECF operates by performing computations on randomly chosen elliptic curves over finite fields. Unlike Fermat's factorization method, which is specifically efficient when factors are close together, ECF is more versatile and effective when n has relatively small prime factors, regardless of their proximity.

How ECF Works

The Elliptic Curve Factorization method works through the following process:

Algorithm 3 Elliptic Curve Factorization Method

```
1: Choose a random elliptic curve $E$ over the ring $\mathbb{Z}_n$
2: Select a random point $P$ on the curve $E$
3: Attempt to perform scalar multiplication $kP$ for a large integer $k$
4: During computation, if an inverse modulo $n$ cannot be computed (i.e., $\gcd(\text{denominator}, n) \neq 1$)
5: Extract the non-trivial factor of $n$ as $\gcd(\text{denominator}, n)$
6: If no factor is found, repeat with a different curve and/or point
```

The key insight is that a failure in modular inversion during point arithmetic reveals a factor of the modulus. This makes ECF particularly effective when n has a relatively small prime factor.

When is ECF a Viable Attack on RSA?

The Elliptic Curve Factorization method (ECF), or Elliptic Curve Method (ECM), is effective when one of the RSA prime factors is relatively small—typically under 200-230 bits. Unlike Fermat's attack, which exploits closeness between p and q, ECF focuses on the *absolute size* of the smaller factor.

- **Best suited for small primes:** If one of the primes is small, ECF can factor the modulus efficiently, regardless of how close p and q are.
- **More general than Fermat's method:** As discussed earlier, Fermat's method works best when $p \approx q$. ECF does not rely on this closeness and remains effective over a wider range of factor pairs.
- **Ineffective for strong keys:** When both p and q are large (e.g., 1024 bits each), ECF becomes infeasible.

Thus, ECF is particularly useful for detecting weak RSA keys where one prime is significantly smaller, even if p and q are not close.

5 Fourier Transform

Definition 3. *The Fourier Transform of a function $f(t)$ is a mathematical transformation that decomposes the function into its constituent frequencies:*

$$F(\omega) = \int_{-\infty}^{\infty} f(t)e^{-j\omega t}\, dt$$

where ω represents angular frequency, j is the imaginary unit $\sqrt{-1}$, and $e^{-j\omega t} = \cos(\omega t) - j\sin(\omega t)$ are the complex exponential basis functions.

The inverse Fourier Transform allows reconstruction of the original signal from its frequency components:

$$f(t) = \frac{1}{2\pi}\int_{-\infty}^{\infty} F(\omega)e^{j\omega t}\, d\omega$$

For practical applications with discrete signals (like digital cryptographic data), the Discrete Fourier Transform (DFT) is used:

$$X_k = \sum_{n=0}^{N-1} x_n e^{-j\frac{2\pi}{N}kn}, \quad k = 0, 1, \ldots, N-1$$

where:

- x_n is the n-th sample of the discrete signal of length N
- X_k is the k-th frequency component

The DFT is computationally expensive, requiring $\mathcal{O}(N^2)$ operations. The Fast Fourier Transform (FFT) algorithm reduces this to $\mathcal{O}(N \log N)$, making it feasible for large datasets.

6 Introduction to Quantum Computing

Quantum computing is an emerging paradigm that leverages the principles of quantum mechanics to process information in fundamentally new ways. Unlike classical computing, which relies on bits to represent data as either 0 or 1, quantum computing introduces the concept of qubits—quantum bits—that enable powerful computational capabilities through unique quantum phenomena.

6.1 Qubits: The Quantum Information Unit

A qubit (just a fancy short for quantum bit) can exist not just in the classical states $|0\rangle$ or $|1\rangle$, but in a superposition of both:

$$|\psi\rangle = \alpha |0\rangle + \beta |1\rangle$$

Here, α and β are complex numbers called *probability amplitudes*, and they determine the likelihood of the qubit being measured in state $|0\rangle$ or $|1\rangle$.

6.2 Superposition of Quantum States

Superposition is essentially the ability of a qubit to simultaneously represent a linear combination of both basis states, $|0\rangle$ and $|1\rangle$. The probabilities of measuring the qubit in each state are given by the squared magnitudes of the amplitudes:

$$|\alpha|^2 + |\beta|^2 = 1$$

This equation ensures that the total probability of all possible outcomes is 1. The state $|\psi\rangle$ is called the *quantum state*, a vector in a 2-dimensional complex Hilbert space. This state captures all information about how a qubit behaves under observation or interaction.

6.3 Quantum Gates: Manipulating Qubit States

Just as classical computers use logic gates (like AND, OR, NOT) to process bits, quantum computers use *quantum gates* to manipulate qubits.
Quantum gates are mathematical operations that change the state of a qubit. They act on the qubit's quantum state vector in its complex vector space, and are typically represented by unitary matrices. These operations are reversible, unlike many classical gates.
For example:

- The **Hadamard gate (H)** creates superposition by transforming a qubit from a definite state ($|0\rangle$ or $|1\rangle$) into an equal combination of both.
- The **Pauli-X gate** is the quantum equivalent of a classical NOT gate; it flips $|0\rangle$ to $|1\rangle$ and vice versa.

Gates are the tools that let us harness and direct the quantum properties of qubits, enabling computation. Without gates, qubits just sit there in superposition. With gates, we build quantum circuits that perform real tasks—leading up to the final step of measurement, where the quantum result is extracted.

6.4 Analogy: Bits vs Qubits

A simple analogy illustrates the distinction:

- A classical bit is like a coin resting on a table—it's either heads (0) or tails (1).
- A qubit in superposition is like a spinning coin in mid-air—it's in a genuine mixture of heads and tails until it lands (i.e., until it's measured).

This analogy captures the essence of quantum uncertainty and parallelism.

6.5 Measurement: From Superposition to Classical State

When a qubit is measured, it collapses from its superposed state into one of the two classical states—$|0\rangle$ or $|1\rangle$. The outcome is random, governed by the probabilities $|\alpha|^2$ and $|\beta|^2$. After measurement, the qubit loses its superposition and remains in the resulting classical state.
The process of measurement is inherently destructive to superposition, and its exact implementation depends on the physical realization of the qubit. For example:

- In the **Superconducting Qubits (used by IBM)**: Measurement is performed using microwave pulses and a resonator that detects the resulting quantum state.

7 Quantum Fourier Transform (QFT)

The Quantum Fourier Transform (QFT) is a fundamental operation in quantum computing, serving as the quantum counterpart to the classical Discrete Fourier Transform (DFT).
In classical computing, the DFT converts a sequence of complex numbers into another sequence that represents the frequency components of the original sequence. Similarly, the QFT operates on the amplitudes of a quantum state, transforming them into a superposition of states with different amplitudes.
Mathematically, for a quantum state $|x\rangle$, the QFT is defined as:

$$\text{QFT}(|x\rangle) = \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} e^{2\pi i x k/N} |k\rangle$$

where $N = 2^n$ for an n-qubit system. This transformation is unitary and reversible, preserving the total probability.

7.1 Understanding the QFT Mechanism

The QFT operates on an n-qubit quantum state $|x\rangle$, transforming it into a superposition of states weighted by complex exponential factors:

$$\text{QFT}(|x\rangle) = \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} e^{2\pi i x k/N} |k\rangle , \quad N = 2^n$$

This transformation is unitary and reversible, preserving the total probability of the quantum system.

7.2 Implementing the QFT in Quantum Circuits

The QFT can be efficiently implemented using a sequence of quantum gates:

- **Hadamard Gates**: Applied to individual qubits to create superpositions.
- **Controlled Phase Shift Gates**: Introduce relative phases between qubits based on their states.
- **Swap Gates**: Reverse the order of qubits to obtain the correct output.

This implementation requires $\mathcal{O}(n^2)$ quantum gates, making it exponentially faster than the classical Fast Fourier Transform (FFT) for large datasets.

7.3 Applications

The QFT is instrumental in several quantum algorithms:

- **Shor's Algorithm**: Utilizes the QFT for efficient integer factorization, which has implications for cryptography.
- **Quantum Phase Estimation**: Employs the QFT to estimate the phase (eigenvalues) of unitary operators, a critical component in various quantum algorithms.

These applications leverage the QFT's ability to analyze periodicity and phase information efficiently.

8 Shor's Algorithm

Shor's Algorithm is a groundbreaking quantum algorithm developed by mathematician Peter Shor in 1994. It efficiently factors large integers, a task that is computationally intensive for classical computers. This capability poses significant implications for cryptography, particularly for encryption methods like RSA that rely on the difficulty of prime factorization.

8.1 Overview

Shor's Algorithm efficiently factors large composite numbers, posing a significant threat to RSA cryptography. It reduces factoring to a period-finding problem, leveraging quantum parallelism for exponential speedup over classical methods. On a powerful quantum computer, it could break RSA by deriving private keys from public keys, undermining its security foundation.

8.2 How Shor's Algorithm Works

The algorithm comprises both classical and quantum components:

Algorithm 4 Shor's Algorithm for Integer Factorization

```
1: Classical Preprocessing:
2:     Select a random integer $a$ such that $1 < a < N$ and $\gcd(a, N) = 1$
3:     If $\gcd(a, N) \neq 1$, a nontrivial factor is found
4: Quantum Order Finding:
5:     Utilize a quantum computer to find the period $r$ of the function $f(x) = a^x \mod N$
6:     This step employs quantum parallelism and the QFT to determine $r$ efficiently
7: Classical Postprocessing:
8:     If $r$ is even and $a^{r/2} \not\equiv -1 \mod N$, compute $\gcd(a^{r/2} \pm 1, N)$ to obtain nontrivial factors of $N$
9:     If conditions are not met, repeat the process with a different $a$
```

8.3 The Role of QFT in Shor's Algorithm

Shor's Algorithm reduces the problem of factoring a composite number N to finding the period r of the function $f(x) = a^x \mod N$, where a is a randomly chosen integer less than N and coprime to it. The QFT is instrumental in determining this period efficiently.

8.4 How QFT Facilitates Period Finding

- **Superposition Creation:** A quantum register is initialized into a superposition of all possible states $|x\rangle$.
- **Function Evaluation:** The function $f(x) = a^x \mod N$ is computed in superposition, entangling the input and output registers.
- **Measurement:** Measuring the output register collapses it to a specific value $f(x_0)$, and the input register becomes a superposition of all x such that $f(x) = f(x_0)$, which are spaced r apart.

- **Quantum Fourier Transform:** Applying the QFT to the input register transforms this periodic superposition into another superposition where the amplitudes are peaked at integer multiples of $1/r$.
- **Measurement and Classical Post-processing:** Measuring the transformed state yields a value that, through continued fraction expansion, allows the extraction of the period r.

This process leverages the QFT's ability to convert periodicity in the time domain into frequency domain information, which is essential for determining r.

8.5 Software Implementation

```python
from qiskit import Aer
from qiskit.algorithms import Shor
from qiskit.utils import QuantumInstance

N = 15
quantum_instance = QuantumInstance(Aer.get_backend('qasm_simulator'), shots
    =100)

shor = Shor(quantum_instance=quantum_instance)
result = shor.factor(N)

print(f"Factors of {N}: {result.factors}")
```

Listing 2: Qiskit Implementation of Shor's Algorithm

This code snippet demonstrates a simple implementation of Shor's Algorithm using Qiskit, a popular quantum computing framework. The algorithm is executed on a quantum simulator, and the factors of the integer $N = 15$ are printed as output.

8.6 Quantum Circuit Representation

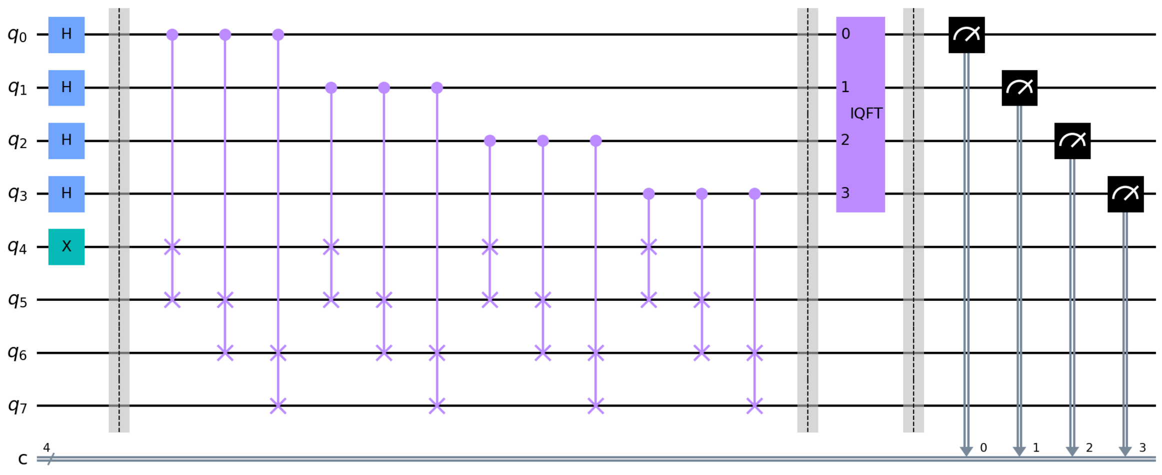

Figure 1: Quantum Circuit Representation of Shor's Algorithm

```python
from qiskit import QuantumCircuit
from qiskit.circuit.library import QFT

def shors_circuit():
    # For factoring N = 15, using a = 2
    # 4 counting qubits, 4 work qubits
    qc = QuantumCircuit(8, 4)

    # Apply Hadamard to counting qubits (0-3)
    qc.h(range(4))

    # Initialize work register (qubits 4-7) to |1>
    qc.x(4)

    qc.barrier()

    # Controlled-U^{2^0}
    qc.cswap(0, 4, 5)
    qc.cswap(0, 5, 6)
    qc.cswap(0, 6, 7)

    # Controlled-U^{2^1}
    qc.cswap(1, 4, 5)
    qc.cswap(1, 5, 6)
    qc.cswap(1, 6, 7)

    # Controlled-U^{2^2}
    qc.cswap(2, 4, 5)
    qc.cswap(2, 5, 6)
    qc.cswap(2, 6, 7)

    # Controlled-U^{2^3}
    qc.cswap(3, 4, 5)
    qc.cswap(3, 5, 6)
    qc.cswap(3, 6, 7)

    qc.barrier()

    # Apply inverse QFT
    qc.append(QFT(4).inverse(), [0, 1, 2, 3])

    qc.barrier()

    # Measure the counting register
    qc.measure(range(4), range(4))

    return qc

# Save circuit as image
qc = shors_circuit()
qc.draw(output='mpl', filename='../media/images/shor_circuit.png')
```

Listing 3: Quantum Circuit for Shor's Algorithm (N=15, a=2)

8.7 Time Complexity Analysis

The overall time complexity of Shor's Algorithm is:

$$\mathcal{O}((\log N)^3)$$

where N is the integer to be factored. This is a polynomial-time complexity, representing an exponential speedup over the best-known classical factoring algorithms. For instance, the General Number Field Sieve (GNFS), a classical algorithm, has a sub-exponential time complexity of approximately:

$$\mathcal{O}\left(e^{(\log N)^{1/3}(\log\log N)^{2/3}}\right)$$

This stark contrast highlights the potential of quantum computing to solve certain problems much more efficiently than classical computing.

8.8 Implications for Cryptography

- **Threat to Data Security:** Encrypted data, including financial transactions and confidential communications, could be decrypted if intercepted and stored until quantum computers become powerful enough to run Shor's Algorithm.

- **Necessity for Quantum-Resistant Cryptography:** The looming threat has accelerated the development of post-quantum cryptographic algorithms designed to withstand quantum attacks. Organizations like the National Institute of Standards and Technology (NIST) are leading efforts to standardize these new cryptographic methods.

9 Quantum Fixed-Point Attacks on RSA

Beyond Shor's algorithm, quantum computing introduces alternative methods to compromise RSA, such as fixed-point attacks. These attacks exploit the mathematical properties of RSA encryption to recover plaintext messages directly from ciphertexts without factoring the modulus n.

Fixed-Point Concept: In RSA, a fixed point is a value M satisfying $M^e \equiv M \mod n$. While such points are rare and typically not useful in classical cryptanalysis, quantum algorithms can efficiently identify these fixed points.

Quantum Algorithm Approach: Researchers have proposed quantum algorithms that utilize quantum Fourier transforms to detect fixed points in RSA encryption. By preparing quantum superpositions and applying modular exponentiation, these algorithms can amplify the probability of measuring a fixed point, thereby recovering the original message M from the ciphertext C without needing to factor n.

Implications: Although these quantum fixed-point attacks are currently theoretical and require quantum computers with substantial capabilities, they highlight potential vulnerabilities in RSA that could be exploited as quantum technology advances. This underscores the importance of transitioning to quantum-resistant cryptographic schemes.

10 Practical Limitations of Quantum Computing

Despite their potential, quantum computers face several significant practical challenges:

1. **Cost and Infrastructure:** Quantum systems are highly expensive to develop and maintain, requiring advanced infrastructure such as electromagnetic shielding and continuous monitoring. These demands currently restrict access to major corporations and research institutions.
2. **Software, Tooling, and Talent Gaps:** Quantum software development remains at an early stage. Although frameworks like Qiskit are emerging, they lack mature toolchains and robust developer ecosystems. There is also a limited pool of talent with quantum computing expertise.
3. **Hardware Limitations and Scalability:** Building quantum systems with large numbers of qubits is critical to solving meaningful problems. However, scaling introduces control complexity, increased noise, and greater cooling requirements.
4. **Qubit Fragility and Decoherence:** Qubits must maintain delicate quantum states like superposition and entanglement, which are extremely sensitive to environmental disturbances such as temperature fluctuations and electromagnetic radiation. This leads to decoherence, one of the primary obstacles in practical quantum computation.

11 Post-Quantum Cryptography Alternatives

- **Lattice-Based:** CRYSTALS-KYBER, DILITHIUM, NTRU, Falcon.
- **Code-Based:** Classic McEliece.
- **Hash-Based Signatures:** SPHINCS+, XMSS, LMS.
- **Multivariate Polynomial:** Rainbow.
- **Isogeny-Based:** SIKE (broken), FrodoKEM.

Feature	**RSA**	**Post-Quantum Alternatives**
Security (Classical)	Good with large keys	Designed to be secure
Security (Quantum)	Vulnerable	Resistant
Key Sizes	Moderate to large	Varies
Signature/Ciphertext Sizes	Moderate	Varies
Performance	Efficient	Varies
Maturity	Well-established	Under research

12 Conclusion

13 Acknowledgements

14 References

15 Roles and Responsibilities

This project was completed through collaborative effort, with each team member making significant contributions in various capacities. The table below outlines the specific roles and responsibilities undertaken by each member of the team.

Team Member	Research Topics	PPT	Report	Simulations
Jigar Fufal	RSA fundamentals and weaknesses, Post-quantum cryptographic alternatives	✓	✓	
Ojas More	Weak RSA exploitation using Fermat's theorem, Close prime factorization using Elliptic Curve Factorization		✓	
Parth Gawande	Fourier Transform theory, Fixed Point Attacks on RSA		✓	✓
Abhishek Mantri	Practical limitations of quantum computing	✓	✓	
Arani Ghosh	Quantum Fourier Transform (QFT) and its application in Shor's Algorithm	✓	✓	
Arka Mukhopadhay	Introduction to quantum computing, qubit theory, quantum measurement theory		✓	✓
Harshit Jain	Shor's Algorithm, Quantum Gates	✓	✓	