Project Plan: Implementing RSA with 1024 and 2048-bit Key Sizes

1. Project Overview

- Briefly discuss the project objectives and requirements.
- Understand the importance of key sizes in RSA security.

2. Understanding RSA Algorithm

- Review the mathematical foundations:
  - Key generation.
  - Encryption and decryption processes.
- Ensure all team members have a solid grasp of RSA mechanics.

3. Decide on Programming Language and Tools

- Options:
  - Python: Easy syntax, int type supports arbitrary precision, pycryptodome library.
  - Java: Built-in BigInteger class.
  - C/C++: High performance, but requires external libraries for big integers (e.g., GMP).
- Decision Factors:
  - Team familiarity.
  - Library support for big integer arithmetic.
  - Ease of implementation and debugging.

4. Implementation Plan

A. Key Generation Module

1. Prime Number Generation
   - Goal: Generate two large prime numbers (p and q) of 512 bits (for 1024-bit RSA) and 1024 bits (for 2048-bit RSA).
   - Methods:
     - Implement probabilistic primality tests (e.g., Miller-Rabin).
     - Use strong random number generators.
   - Tasks:
     - Research and select appropriate algorithms.
     - Implement prime generation function.

2. Compute Modulus (n) and Totient ($\phi(n)$)
   - Compute n: $n = p * q$
   - Compute $\phi(n)$: $\phi(n) = (p - 1) * (q - 1)$

3. Select Public Exponent (e)
   - Common choice is e = 65537 for its properties.
   - Ensure $\gcd(e, \phi(n)) = 1$.

4. Compute Private Exponent (d)
   - Calculate the modular inverse: $d \equiv e^{-1} \bmod \phi(n)$
   - Use the Extended Euclidean Algorithm.

5. Key Storage
   - Decide on a format for storing keys (e.g., PEM format).
   - Ensure private keys are stored securely.

B. Encryption Function

1. Message Preparation
   - Convert plaintext to integer m, where $0 \leq m < n$.
   - Consider implementing padding schemes (e.g., PKCS#1) to enhance security.

2. Encryption Process

- Compute ciphertext: c = m^e mod n

C. Decryption Function

1. Decryption Process
   - Recover plaintext: m = c^d mod n

2. Message Recovery
   - Convert integer m back to plaintext.
   - Remove padding if used.

5. Testing and Validation

- Unit Tests:
  - Test each module individually (prime generation, key generation, encryption, decryption).
- Integration Tests:
  - Test the complete flow with known inputs and outputs.
- Performance Tests:
  - Measure execution time for key generation, encryption, and decryption.
- Edge Cases:
  - Test with boundary values (e.g., very small messages, maximum message size).

6. Optimization

- Algorithm Optimization:
  - Use efficient algorithms for modular exponentiation (e.g., Montgomery multiplication).
- Code Optimization:
  - Profile code to identify bottlenecks.
  - Optimize prime generation and exponentiation steps.
- Library Utilization:
  - Leverage optimized libraries for big integer operations.

7. Documentation

- Code Documentation:
  - Comment code thoroughly.
  - Explain complex algorithms and decisions.
- Project Report:
  - Document the implementation process.
  - Include explanations of algorithms used.
  - Discuss any challenges and how they were overcome.

8. Timeline and Milestones

- Week 1:
  - Finalize programming language and tools.
  - Set up development environment.
  - Begin prime number generation module.
- Week 2:
  - Complete prime generation.
  - Implement key generation module.
- Week 3:
  - Develop encryption and decryption functions.
  - Start testing individual modules.
- Week 4:
  - Perform integration testing.
  - Optimize code.
  - Begin documentation.

- Week 5:
  - Finalize documentation.
  - Prepare presentation (if required).
  - Buffer time for any unforeseen issues.

Summary

By following this plan, our team can systematically approach the RSA implementation project. During tonight's meeting, we'll discuss each point, make decisions where needed, and ensure everyone is clear on their responsibilities. The goal is to collaborate effectively, leverage each other's strengths, and successfully implement RSA encryption and decryption with 1024 and 2048-bit key sizes.