



Supervisor: Prof. Mohammed M. Abu Shquier

Mohammad Hiary – 221697: social engineering

Karim Amjad Khattab – 222269: AES: Analyzer the Role and Impact of Ransomware Encryption

Ahmad Eyad AL Wheidi – 221237: RSA: Analyzer The Role and Impact of Ransomware Encryption

Hazem Khaled Ahmed Al-Tamimi – 220039 : Basic information



Week: 1

Date: [5/11/2025]

1. Objective

The main objective of this week was to study and implement an efficient primality testing algorithm as part of the cryptographic foundations of the project. I chose to work on the Miller Rabin probabilistic primality test, which is widely used in modern encryption algorithms such as RSA for verifying large prime numbers.

2. Work Done

During this week, I successfully implemented the `is_probable_prime()` function in Python. The function determines whether a given integer is probably prime using the Miller-Rabin test with multiple random iterations for accuracy.

I also analyzed the code line by line to understand the mathematical and logical operations behind each part. This included:

- Handling small prime checks for quick filtering.
- Decomposing $n - 1$ into the form $d \times 2^s$.
- Testing multiple random bases (a) to verify whether the number behaves like a prime under modular exponentiation.
- Ensuring probabilistic accuracy through 64 random rounds, significantly reducing the chance of error.

3. Implementation Summary

The function follows these main steps:

1. Input validation: Reject numbers smaller than 2.
2. Small prime check: Quickly identify small prime numbers.
3. Factorization: Express $n - 1$ as $d \times 2^s$ to prepare for modular testing.
4. Miller-Rabin rounds: Randomly choose bases a to test whether n passes the conditions of a probable prime.
5. Output: Return True if no base proves compositeness after all rounds, otherwise False.

This process provides a fast and reliable way to test large numbers used in cryptographic systems.

4. Results

- The function was tested with various values such as 97 (prime) and 91 (composite).
- Results were consistent with expected outcomes:

`is_probable_prime(97) → True`

`is_probable_prime(91) → False`

- The algorithm performed efficiently even for large integers, confirming the effectiveness of the Miller-Rabin approach.

5. Challenges Faced

- Understanding how to correctly decompose $n - 1$ into d and s .
- Ensuring random base generation stayed within the correct range $[2, n - 2]$.
- Interpreting the modular arithmetic logic to correctly identify “witnesses” of compositeness.

→ I overcame these challenges by carefully analyzing the mathematical steps to decompose $n - 1$ into d and s , validating the random base generation within the range $[2, n - 2]$, and verifying modular arithmetic results through iterative testing to correctly identify witnesses of compositeness.

6. Next Steps (Week 2 Plan)

- Integrate this function into a hybrid cryptography system (RSA + AES) for secure key generation.
- Begin testing with larger primes to evaluate performance and accuracy.
- Document time complexity and compare it with deterministic primality tests.

7. Conclusion

This week focused on the research, coding, and verification of the Miller-Rabin primality test. The implementation works correctly and forms a strong mathematical base for upcoming cryptographic development. The algorithm ensures probabilistic certainty suitable for real-world encryption systems like RSA.

```
graph TD
    Start([Start]) --> Input[Input n, Rounds]
    Input --> Cond1{If n < 2: return: False}
    Cond1 --> Cond2{If n is even: Return False}
    Cond2 --> Write[Write n-1 = d * 2^s]
    Write --> Repeat[Repeat Rounds times]
    Repeat --> Pick[Pick Random a in [2, n-2]]
    Pick --> Calc1[Calculate x = a^d mod n]
    Calc1 --> Cond3{If x == 1 or x == n-1? (yes)}
    Cond3 --> Calc2[Calculate r = r + 1]
    Calc2 --> Cond4{While r < s}
    Cond4 --> Calc3[Calculate x = x^2 mod n]
    Calc3 --> Cond5{Yes next round}
    Cond5 --> Repeat
    Cond3 --> NO((NO))
    NO --> Calc2
    Cond4 --> Cond6{Return False}
    Cond6 --> Composite((Composite))
    Cond6 --> End[End of all rounds]
    End --> ReturnTrue[Return True]
    ReturnTrue --> ProbablePrime((Probably Prime))
```