Name	**Muhammad Anas Hassaan**
CMS_ID	**410555**
Instructor	**Sir Alamgir**

Assignment #4

---

1. Pseudocode and Algorithm Design

Approach A: Basic (Textbook) RSA

Key Generation:

1. Choose two large prime numbers p and q
2. Compute $n = p * q$
3. Compute $\varphi(n) = (p-1)*(q-1)$
4. Choose e such that $1 < e < \varphi(n)$ and $\gcd(e, \varphi(n)) = 1$
5. Compute $d \equiv e^{-1} \bmod \varphi(n)$
6. Public Key: (e, n), Private Key: (d, n)

Encryption:

1. Convert plaintext M to integer m
2. Compute ciphertext c = m^e mod n

Decryption:

1. Compute m = c^d mod n
2. Convert m back to plaintext M

Approach B: CRT-Based RSA

Decryption (Optimized):

1. Precompute dp = d mod (p-1), dq = d mod (q-1), qinv = q^{-1} mod p
2. Compute m1 = c^dp mod p
3. Compute m2 = c^dq mod q
4. h = qinv * (m1 - m2) mod p
5. Compute m = m2 + h * q

2. Computational Complexity Analysis

Operation	**Basic RSA**	**CRT-Optimized RSA**
Key Generation	$O(\log^2 n)$	Same
Encryption	O(log e)	Same
Decryption	O(log d)	~4x faster using CRT

CRT Speedup: Decryption with CRT is ~4x faster due to reduced operand size.

3. Implementation Overview (Python)

Key Modules Used:

- Crypto.Util.number for prime generation and inverse
- pow(a, b, c) for efficient modular exponentiation

Key Generation:

```
p = getPrime(512)
q = getPrime(512)
n = p * q
phi = (p - 1) * (q - 1)
e = 65537
d = inverse(e, phi)
```

Encryption/Decryption:

```
c = pow(m, e, n)
m = pow(c, d, n)
```

CRT Decryption:

```
dp = d % (p - 1)
dq = d % (q - 1)
qinv = inverse(q, p)
m1 = pow(c, dp, p)
m2 = pow(c, dq, q)
h = (qinv * (m1 - m2)) % p
m = m2 + h * q
```

4. Attack Simulations

Low Public Exponent Attack (Håstad's Attack):

- Use e = 3, encrypt same message for 3 recipients.
- Use CRT to combine ciphertexts.
- Extract plaintext with cube root.

```
# Simulate CRT + nth root
m = int(nth_root(c_combined, 3))
```

Wiener's Attack:

- Use d small relative to n.

- Use continued fraction attack (via wiener_attack.py).
- Successful if $d < n^{0.25}$.

Chosen Ciphertext Attack:

- Let $c' = c * r^e \mod n$, decrypt c'.
- Factor out r from result.

5. Evaluation and Results

Metric	**Basic RSA**	**CRT-Based RSA**
Decryption Time	High	Significantly lower
Security	Vulnerable	Same
Resilience	Weak with reuse	Same
Overall	Slower, simpler	Faster, needs care

6. Conclusion

1. CRT-Based RSA is faster, especially in decryption.

2. Both implementations are vulnerable without padding.

3. Adding padding (e.g., OAEP) is recommended for security.

4. Attack simulations confirm textbook vulnerabilities.