RSA Assignment

Sujoy Maity
Course: Cryptographic & Security Implementations

Submission Deadline: 12 August 2025

Graded Assignment
Date: August 11, 2025

Problem Statement

The objective of this graded assignment is to implement the RSA public-key cryptographic algorithm using the C programming language on a Linux-based operating system. The focus is on measuring the performance of RSA key generation, encryption, and decryption in terms of clock cycles. The GNU MP (GMP) library's `mpz_t` data type is used for large integer arithmetic. The implementation must be compiled using the `gcc` compiler.

Assignment Tasks (Step-wise)

1. **Step 1: Prime Number Generation**
   - Generate two large primes p and q, each of 512 bits.
   - Repeat the process 1,000,000 times.
   - Record min, max, and average clock cycles for both primes.
2. **Step 2: RSA Modulus and Euler's Totient**
   - **2a:** Compute RSA Modulus $N = p \times q$
   - **2b:** Compute Euler's Totient $\phi(N) = (p-1)(q-1)$
   - Record clock cycles for both.
3. **Step 3: Private Key Generation**
   - Use $e = 2^{16} + 1 = 65537$ as public exponent.
   - Compute private key d such that $e \cdot d \equiv 1 \mod \phi(N)$
   - Record clock cycles.
4. **Step 4: Message Encryption and Decryption**

- **4a:** Generate a 1023-bit message m
- **4b:** Encrypt: $c = m^e \mod N$
- **4c:** Decrypt: $m' = c^d \mod N$
- **4d:** Verify $m' = m$

5. **Step 5: Repeat for Other Key Sizes**
   - Repeat Steps 1–4 for:
     - 768-bit primes
     - 1024-bit primes

System Specifications

- **CPU:** AMD Ryzen 7 7435HS (8 Cores, 16 Threads, x86_64)
- **RAM:** 7.66 GB DDR5
- **Operating System:** Ubuntu 24.04.3 LTS (64-bit)
- **Compiler:** `gcc (Ubuntu 13.3.0-6ubuntu2~24.04) 13.3.0`
- **Library Used:** GNU Multiple Precision Arithmetic Library (GMP)
- **PRG Used:** GMP Mersenne Twister (`gmp_randinit_mt`)

Outputs and Results

512-bit Key Size

```
================================================
    RSA TESTING WITH 512-BIT PRIME NUMBERS
================================================
Pseudo-Random Generator (PRG): GMP Mersenne Twister (gmprandinitmt)
Generating 512-bit prime pairs for 1000000 iterations...

=== Prime Generation Timing for 512-bit Primes ===
```

Listing 1: All Steps Output for 512-bit RSA

```
Prime p:
  Min cycles:    1691236
  Max cycles:    57781706
  Avg cycles:    3932649.86
Prime q:
  Min cycles:    1696102
  Max cycles:    103752227
  Avg cycles:    3931185.63

Final prime p:
1074846536975741729748272538052759524141166897704 9...0299631809702458462260008107390995161
    ↪ 1179487020591  [total digits: 155]
```

```
Final prime q:
10258661680180812005043536310180886040362727424739...829862445100674320930568374425398607
    ↪ 8237966372633  [total digits: 155]

Step 2a: N Computation Timing (N = p x q)
  Min cycles:    93
  Max cycles:    91016
  Avg cycles:    117.33
RSA Modulus N:
11026486980948089930010204883066766339951170838398...359177540841789388103392646023554049
    ↪ 5744849886103  [total digits: 309]

Step 2b: phi(N) Computation Timing (phi(N) = (p-1)(q-1))
  Min cycles:    124
  Max cycles:    129487
  Avg cycles:    145.88
Euler's Totient phi(N):
11026486980948089930010204883066766339951170838398...499351914770869220946823460859055925
    ↪ 6327396492880  [total digits: 309]

Step 3 (Private key generation): 276644 cycles
Private key d:
61341637371009721341874370177415999204766732929059...295647824292096387178238804941640905
    ↪ 0919488971633  [total digits: 308]

Step 4a (Message generation 1023-bit): 1302 cycles
Original 1023-bit message (m):
51124955696437923345067561748249051488228635168820...110385559812824837821203303122617248
    ↪ 0638869694953  [total digits: 308]

Step 4b (Encryption): 20429 cycles
Encrypted message (c):
50631889249617918755419787630501883442209360162803...80197552431414526394523510676092068
    ↪ 7175344762033  [total digits: 308]

Step 4c (Decryption): 818431 cycles
Decrypted message (m'):
51124955696437923345067561748249051488228635168820...110385559812824837821203303122617248
    ↪ 0638869694953  [total digits: 308]

Step 4d (Message verification): 155 cycles
Message verification:  SUCCESS
```

768-bit Key Size

```
================================================
    RSA TESTING WITH 768-BIT PRIME NUMBERS
================================================
Pseudo-Random Generator (PRG): GMP Mersenne Twister (gmprandinitmt)
Generating 768-bit prime pairs for 10000000 iterations...

=== Prime Generation Timing for 768-bit Primes ===
```

Listing 2: All Steps Output for 768-bit RSA

```
Prime p:
  Min cycles:    4734506
  Max cycles:    354288353
  Avg cycles:    13926892.66
Prime q:
  Min cycles:    4735963
  Max cycles:    277758766
  Avg cycles:    13950695.01

Final prime p:
1303331785510581713437814683506451822945217349524 0...59843204403288047803383390094189496938542403283357  [total digits: 232]

Final prime q:
1122962938907303366625639948227733054284658250799 9...22456039714439641198619065040693812116304869838699  [total digits: 232]

Step 2a: N Computation Timing (N = p x q)
  Min cycles:    248
  Max cycles:    455080
  Avg cycles:    281.28
RSA Modulus N:
1463593292228265987835159705320521199985232658560 9...78222002585984952370505665824644558580856581232543  [total digits: 463]

Step 2b: phi(N) Computation Timing (phi(N) = (p-1)(q-1))
  Min cycles:    279
  Max cycles:    427335
  Avg cycles:    312.45
Euler's Totient phi(N):
1463593292228265987835159705320521199985232658560 9...95922758468257263368503210689761249526009308110488  [total digits: 463]

Step 3 (Private key generation): 120342 cycles
Private key d:
9212305552238918513845335154825033190501675755420 1...42541002938402434826344293210908972095112836548297  [total digits: 462]

Step 4a (Message generation 1023-bit): 2511 cycles
Original 1023-bit message (m):
2131402450687055553646997973313533053871743085395 5...43571714505047313952569703655858698677129122659132  [total digits: 307]

Step 4b (Encryption): 47337 cycles
Encrypted message (c):
8471173949094491861650231533753056635694590766886 6...36627052260395231128777332357417885255495927657637  [total digits: 462]

Step 4c (Decryption): 3196038 cycles
Decrypted message (m'):
2131402450687055553646997973313533053871743085395 5...43571714505047313952569703655858698677129122659132  [total digits: 307]

Step 4d (Message verification): 217 cycles
Message verification:  SUCCESS
```

1024-bit Key Size

```
================================================
    RSA TESTING WITH 1024-BIT PRIME NUMBERS
================================================
Pseudo-Random Generator (PRG): GMP Mersenne Twister (gmprandinitmt)
Generating 1024-bit prime pairs for 100000 iterations...

=== Prime Generation Timing for 1024-bit Primes ===
```

Listing 3: All Steps Output for 1024-bit RSA

```

Prime p:
  Min cycles:    10057640
  Max cycles:    324425615
  Avg cycles:    33810878.24
Prime q:
  Min cycles:    10071589
  Max cycles:    309426748
  Avg cycles:    33714284.99

Final prime p:
126349267987442980046978453670807531255007005939 43...5413822681638865873766932748329470721
    ↪ 3850636238213  [total digits: 309]

Final prime q:
159495803348172581285139332993200329929168047690 38...8241170489035913509205622821551716929
    ↪ 1977690549829  [total digits: 309]

Step 2a: N Computation Timing (N = p x q)
  Min cycles:    372
  Max cycles:    84103
  Avg cycles:    413.17
RSA Modulus N:
201521780001107627979816501433303446920409259067 18...2845366319571215785581381136572554500
    ↪ 2940190415577  [total digits: 617]

Step 2b: phi(N) Computation Timing (phi(N) = (p-1)(q-1))
  Min cycles:    403
  Max cycles:    99479
  Avg cycles:    442.87
Euler's Totient phi(N):
201521780001107627979816501433303446920409259067 18...9190373148896436402608825566691366849
    ↪ 7111863627536  [total digits: 617]

Step 3 (Private key generation): 112685 cycles
Private key d:
102069273316703031923371972299267201993928085592 50...0112230439362013976047078075907551935
    ↪ 6899632287905  [total digits: 617]

Step 4a (Message generation 1023-bit): 1364 cycles
Original 1023-bit message (m):
826367649709047759966182307817780991832936384992 35...5563749742332897446509395312228213498
    ↪ 8392694573648  [total digits: 308]

Step 4b (Encryption): 76880 cycles
Encrypted message (c):
```

```
171780962519246193924664807611504291046392614845281...10022182764581382131682808368181505801361735791676  [total digits: 617]

Step 4c (Decryption): 6375398 cycles
Decrypted message (m'):
826367649709047759966182307817780991832936384992351...5563749742332897446509395312228213498392694573648  [total digits: 308]

Step 4d (Message verification): 186 cycles
Message verification:  SUCCESS
```

Performance Comparison Table

Table 1: Clock Cycle Comparison Across Key Sizes (Updated with New Measurements)

Category	Metric	512-bit	768-bit	1024-bit
Prime p	Min	1,691,236	4,734,506	10,057,640
	Max	57,781,706	354,288,353	324,425,615
	Avg	3,932,649.86	13,926,892.66	33,810,878.24
Prime q	Min	1,696,102	4,735,963	10,071,589
	Max	103,752,227	277,758,766	309,426,748
	Avg	3,931,185.63	13,950,695.01	33,714,284.99
RSA Modulus (N)	Min	93	248	372
	Max	91,016	455,080	84,103
	Avg	117.33	281.28	413.17
Euler's Totient (phi(N))	Min	124	279	403
	Max	129,487	427,335	99,479
	Avg	145.88	312.45	442.87
Private Key (d)		276,644	120,342	112,685
Message Generation		1,302	2,511	1,364
Encryption		20,429	47,337	76,880
Decryption		818,431	3,196,038	6,375,398
Verification		155	217	186

Observations

- **Prime Generation:** The time to generate primes p and q increases significantly with key size. For instance, the average cycle count for p generation increases from 3.93 million (512-bit) to 13.93 million (768-bit) and 33.81 million (1024-bit). This is expected, as larger primes require more iterations in the primality testing process.
- **RSA Modulus (N) Computation:** The multiplication of p and q to obtain N is extremely fast compared to prime generation. Even at 1024 bits, the average time is only 413 cycles, showing that this step contributes negligibly to the overall RSA setup time.

- **Euler's Totient ($\phi(N)$) Computation:** Similar to N computation, $\phi(N)$ calculation is very fast, with an average of 442 cycles for the 1024-bit case. This is due to its simple arithmetic structure involving $(p-1)(q-1)$.
- **Private Key Generation:** Unlike most steps, the time required for computing the private key d does not strictly increase with key size. Interestingly, the cycle count decreases slightly from 120,342 cycles (768-bit) to 112,685 cycles (1024-bit), possibly due to variations in prime characteristics or system-level timing fluctuations during modular inverse computation.
- **Encryption:** The encryption step shows moderate cycle counts, ranging from 20,429 cycles (512-bit) to 76,880 cycles (1024-bit), scaling proportionally with key size as expected due to larger modular exponentiations.
- **Decryption:** Decryption is by far the most time-consuming operational step, with cycle counts in the millions (e.g., 6.37 million for 1024-bit). This is due to the large exponent size of d in the modular exponentiation.
- **Verification:** The verification step is extremely fast in all cases, consistently taking fewer than 220 cycles regardless of key size.

Conclusion

The performance evaluation of the RSA implementation using GMP shows that prime generation dominates the computational cost, with significant variation in clock cycles between the minimum and maximum cases. As the key size increases, the average clock cycles for prime generation and modulus computation increase, while the private key computation (d) shows a counter-intuitive decrease in cycles—likely due to the fixed public exponent ($e = 65537$) simplifying the modular inverse calculation. Encryption is consistently fast, whereas decryption is more expensive due to the large private exponent. These results align with typical RSA behavior, with the exception of the private key generation trend, which may be attributed to the mathematical properties of the selected exponents rather than an algorithmic improvement.

References

[1] OpenAI. *ChatGPT v5* — interactive assistance for code-related issues, suggestions, and relevant information.

[2] Overleaf Documentation and Stack Overflow Community. Resources and Q&A used for LaTeX coding support and troubleshooting.