RSA Assignment Problem Statement

Cryptographic & Security Implementations

Graded Assignment

Submission Deadline: **12 August 2025**

August 7, 2025

Objective

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 will be 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 must be used for large integer arithmetic. The implementation must be compiled using the `gcc` compiler.

Assignment Tasks

The assignment is divided into several clearly defined steps:

Step 1: Prime Number Generation

- Generate two large prime numbers p and q, each of 512 bits.
- Use GMP's `mpz_t` functions to generate these primes uniformly at random.
- Repeat the prime generation process one million (1,000,000) times.
- For each iteration, record the number of clock cycles taken to generate the primes.
- At the end, compute and print:
  - Minimum number of clock cycles
  - Maximum number of clock cycles
  - Average number of clock cycles

Step 2: Compute RSA Modulus and Euler's Totient

- Compute $N = p \times q$
- Compute $\phi(N) = (p - 1) \times (q - 1)$
- Record the number of clock cycles taken to compute these values

Step 3: Public and Private Key Generation

- Choose a fixed public exponent $e = 2^{16} + 1$
- Compute the private key d such that $e \cdot d \equiv 1 \mod \phi(N)$
- Record the number of clock cycles required to compute d

Step 4: Message Encryption and Decryption

- Prepare a message m of 1023 bits.
- Encrypt the message: $c = m^e \mod N$
- Decrypt the ciphertext: $m' = c^d \mod N$
- Verify that the decrypted message m' matches the original message m

Step 5: Repeat with Larger Key Sizes

Repeat the entire process (Steps 1–4) using:

- 768-bit primes for p and q
- 1024-bit primes for p and q

This will result in three datasets corresponding to key sizes:

- 512-bit primes
- 768-bit primes
- 1024-bit primes

Implementation Requirements

- Programming Language: C
- Operating System: Any Linux-based OS
- Compiler: `gcc`
- Library: GMP (GNU Multiple Precision Arithmetic Library)
- Data Type: Use `mpz_t` for all big integer computations
- Performance Measurement: Record the number of clock cycles for all computational steps

System Information

Include the following system specifications in your final report:

- CPU model and specifications
- RAM size and type
- Operating System version
- Compiler version (output of `gcc --version`)

Submission Guidelines

- Submit a complete and working source code with appropriate comments.
- Submit a report (in PDF) prepared using LaTeX (Overleaf or offline) that includes:
  - Problem statement (this document)
  - Output datasets and analysis
  - System specifications
  - Observations
- Mention any external sources, libraries, or documentation you took help from.
- **Submission Deadline: 12 August 2025**
- **Submit the report and code in the class group before the deadline.**

RSA Assignment Outputs

Ankan Ghosh (CrS2402)
Cryptographic Security and Implementations

August 11, 2025

1 Terminal output for 512-bit primes and 1 million runs

```
Progress: [##########] 100% Finished!
Over 1000000 trials (PRIME_BITS=512):

Step-1:
Prime p generation: min=1307552, max=753311190, avg=4543347.69
Prime q generation: min=1307818, max=554238336, avg=4552738.71

Step-2:
n computation:      min=1060, max=17927778, avg=1890.64
phi computation:    min=1330, max=11219194, avg=2012.68

Step-3:
Clock cycles for computing d: 15054

Step-4:
Starting encryption...
Encryption done. Clock cycles for encryption:  32916
Starting CRT decryption...
CRT decryption done. Clock cycles for CRT decryption: 312390
CRT decryption verified. Decrypted message matches with original message!

Example run:
Message (hex):     679e0be3645080a6aa42e4881eefdd0c51376a0d3a9fc350dba68f24560c4
↪  55bb636d3da6ca655a3de2ee13363b9998449bd1f0fad9f859650af6fd075575ab99dc570db4
↪  11b36139c1d1f7a856189bc23e3d5270c936a766575cb612e14a08d84e4f02c27dc3ed6eab60
↪  886dc7e71f09fb8fd10e4beec64087af767ab682aa4

Encrypted (hex):   60b5906a9bc6e7fd33ef5da62e6cecb09eb5b4ca2809604d68267660a0dfa
↪  638bfb6f3d47e8721bbb457e265b2954e5ec15e5d71d3e6495e464d27dfabdd4c9e613a1319a
↪  38394d6ec25b321e392fe0811aeb695763090dc9ec4e58cef7c7d1d1af95941f052d7cc0b973
↪  327d30da87cbd8a0cf1086d7662cb9e81f639394ea0
```

```
Decrypted (CRT):   679e0be3645080a6aa42e4881eefdd0c51376a0d3a9fc350dba68f24560c4
↪  55bb636d3da6ca655a3de2ee13363b9998449bd1f0fad9f859650af6fd075575ab99dc570db4
↪  11b36139c1d1f7a856189bc23e3d5270c936a766575cb612e14a08d84e4f02c27dc3ed6eab60
↪  886dc7e71f09fb8fd10e4beec64087af767ab682aa4
```

2 Terminal output for 768-bit primes and 100,000 runs

```
Progress: [##########] 100% Finished!
Over 100000 trials (PRIME_BITS=768):

Step-1:
Prime p generation: min=3690156, max=1949028690, avg=25339789.30
Prime q generation: min=3693816, max=1728135366, avg=25202711.55

Step-2:
n computation:      min=1244, max=2124740, avg=4525.70
phi computation:    min=1530, max=7878750, avg=4323.43

Step-3:
Clock cycles for computing d: 15876

Step-4:
Starting encryption...
Encryption done. Clock cycles for encryption:  103272
Starting CRT decryption...
CRT decryption done. Clock cycles for CRT decryption: 1192934
CRT decryption verified. Decrypted message matches with original message!

Example run:
Message (hex):     41199860d091364d8951b4a18c7621adefcf90c9b8a5db2282f7038760b65
↪  c47e7d743648cdaecbf6f7f6670f7195f0d88f75afb38d504fc60f74056c357e93f99d8d7e71
↪  57b8757c82eec739db05a7773ffaecd5de0dd20beaec59b3336308ce4c4d657dadf8969813b2
↪  ced3b8c18e1537fb542f8b64bbfca590f8e31f3f526

Encrypted (hex):   aa2a8a6a875a8eabc6110e06b0ac4beb556f7c54aa221c90f3d6a147f7909
↪  fe593764b692e207c880986a9cead5a1b1d9e3972ff387692d96c638b1d6f865fdd06d97c146
↪  61d7a5701a780dcbdaca2fe3a8fbf390edd382180105580cc39ea23353a1a3f7cddbd62873b8
↪  bb459d77a211646c63de00e45b34a402c2fbea7ebd89c7e6f5372525805f42d587b76e03db18
↪  375464bd52a64283659d9d74bee6bd1b3ad78adab7fc99b3e2948f4dd3829f2933fbdb4a2b02
↪  c91907618878869c07f

Decrypted (CRT):   41199860d091364d8951b4a18c7621adefcf90c9b8a5db2282f7038760b65
↪  c47e7d743648cdaecbf6f7f6670f7195f0d88f75afb38d504fc60f74056c357e93f99d8d7e71
↪  57b8757c82eec739db05a7773ffaecd5de0dd20beaec59b3336308ce4c4d657dadf8969813b2
↪  ced3b8c18e1537fb542f8b64bbfca590f8e31f3f526
```

3 Terminal output for 1024-bit primes and 100,000 runs

```
Progress: [##########] 100% Finished!
Over 100000 trials (PRIME_BITS=1024):

Step-1:
Prime p generation: min=7809058, max=2900535724, avg=65508347.34
Prime q generation: min=7800828, max=3235912608, avg=65201877.44

Step-2:
n computation:      min=1518, max=8118670, avg=5532.94
phi computation:    min=1832, max=1746384, avg=4922.27

Step-3:
Clock cycles for computing d: 10998

Step-4:
Starting encryption...
Encryption done. Clock cycles for encryption:  5690334
Starting CRT decryption...
CRT decryption done. Clock cycles for CRT decryption: 2968810
CRT decryption verified. Decrypted message matches with original message!

Example run:
Message (hex):     4efddedb79a2876ce432ef2c90a6c90ae351850b2cc639d83c3bf586d082b
↪  eeafdd752a6500f567e23874cb158b11a2be94960864039eb018b02942caf7a1eab821a33fec
↪  45a367dd2db76848ce7a484dbd9054e5fb2c819dce9589adf190f578f08b63077a046ebaa47f
↪  3416f45016ec39c590876c1acf68060f3f17c0509d1

Encrypted (hex):   3d49e8dc6f20a0130bfa1c9d18677d1b3df90d4c1d90c35152b553bc365e4
↪  37b0a9cdfd4aba8fd68ed27824c46752734ea9cfab8de58f6e77b73fcda826d1aa7db6f41f24
↪  e08a62bac7e898c8286542b5c0a1f25d6719636bb998ef027cf119681e3398301073861f3675
↪  2a333fb40efe888fc211db618e7ceb339f267d357ecd54c0da8e9bab97620aab7bac3a6eb014
↪  d1e440f547ef5fc472e32b95722c50c18ae76b65b999dcb3e48cabb02177382614dc2e741787
↪  2b33ccfe1d12d2a078f8c24a0efb26739bf3f6d0b1860c9370c836b7406d1263b5e819704571
↪  c2e552012f9ac9e91cc22c0cb6d85bb218f44267be524fe1eb3d6cc7f863fb9d8f15aab

Decrypted (CRT):   4efddedb79a2876ce432ef2c90a6c90ae351850b2cc639d83c3bf586d082b
↪  eeafdd752a6500f567e23874cb158b11a2be94960864039eb018b02942caf7a1eab821a33fec
↪  45a367dd2db76848ce7a484dbd9054e5fb2c819dce9589adf190f578f08b63077a046ebaa47f
↪  3416f45016ec39c590876c1acf68060f3f17c0509d1
```

4 Observations

In my device, running 1 million trials for 512-bit prime generations took about **1 hour 20 mins**. Running 100,000 trials for 768-bit prime took approximately **1 hour** time. Finally, I ran 100,000 trials for 1024-bit prime generations and that took me about **1 hr 35 mins**.

The computation for **n** and **phi** does not take much time on average, whereas computation for **d** takes relatively more clock cycles. Prime generations take the most time, which increases with size of the prime bits.

One important thing we can observe right away by looking at the terminal outputs is that *the encrypted message length can be bigger than that of the original message length!* But even then, the decrypted message is same as the original plaintext.

5 System Information

5.1 CPU Model and Specifications:

- **Processor -** 11th Gen Intel(R) Core(TM) i3-1115G4 @ 3.00GHz (3.00 GHz)
- **System type -** 64-bit operating system, x64-based processor
- **Graphics Card -** 128 MB, Intel(R) UHD Graphics

5.2 RAM size and type:

Installed RAM: 8.00 GB (7.74 GB usable) DDR4

5.3 Operating System version:

Ubuntu 24.04.1 LTS (Noble Numbat) (64-bit) running on VirtualBox

5.4 Compiler version:

gcc (Ubuntu 13.3.0-6ubuntu2 24.04) 13.3.0

6 External Sources, Libraries and Documentations

- GMP library <gmp.h>, which is a free library for arbitrary precision arithmetic, operating on signed integers, rational numbers, and floating-point numbers. <x86intrin.h> library for reading timestamp counters: $\_\_rdtsc()$ for cycle-accurate timing. Also, libraries like <math.h>, <time.h>, <stdio.h>, <stdlib.h>, <unistd.h>.
- **ChatGPT v5 by OpenAI** for code-related issues and other relevant suggestions and information.
- **Wikipedia** for information regarding Miller-Rabin primality testing, GNU MP (GMP) library, RSA algorithm in general and other relevant stuffs.
- **Overleaf documentations** and **Stack Overflow** for aid related to latex coding.