**PARALLEL PROCESSING PROJECT**

***“****RSA cracking using Prime Factorization****”***

Samuel Wilson (149105006)

Varun Chaudhary (149105452)

**ABSTRACT** –

**RSA** is a [*public-key* cryptosystems](https://en.wikipedia.org/wiki/Public-key_cryptography) and is widely used for secure data transmission. In RSA, the [encryption key](https://en.wikipedia.org/wiki/Encryption_key) is public and differs from the [decryption key](https://en.wikipedia.org/wiki/Decryption_key) which is called as the “secret key“. Therefore, it is an *asymmetric* cryptosystem. This asymmetry is based on the practical difficulty of [factoring](https://en.wikipedia.org/wiki/Factorization) the product of two large [prime numbers](https://en.wikipedia.org/wiki/Prime_number) (*prime factorization*), the [factoring problem](https://en.wikipedia.org/wiki/Factoring_problem).

A user of RSA creates and then publishes a public key based on the product of two large [prime numbers](https://en.wikipedia.org/wiki/Prime_number), along with an auxiliary value. The prime numbers must be kept secret. Anyone can use the public key to encrypt a message, but with currently published methods, if the public key is large enough, only someone with knowledge of the prime numbers can feasibly decode the message. The **RSA Problem** consists of *cracking the public key using prime factorization since the public key is an extremely large value and hence, impractical to factorize.*

The public key is generated with the following procedure:

1. Choose two distinct [prime numbers](https://en.wikipedia.org/wiki/Prime_number) ‘*p’* and ‘*q’*.
   * For security purposes, the integers *p* and *q* should be chosen at random, and should be similar in magnitude but 'differ in length by a few digits' to make factoring harder.
2. Compute *n* = *pq*.
   * *‘n’* is used as the [modulus](https://en.wikipedia.org/wiki/Modular_arithmetic) for both the public and private keys. Its length, usually expressed in bits, is the [key length](https://en.wikipedia.org/wiki/Key_length).

**IMPLEMENTATION** –

The impractical scenario of factorizing a very large public key is carried out using *OpenMP nested parallelisation*. The key is fetched as input and threads are used to parallelise the cracking procedure. For finding out the prime factors, the code has a ready-made list of prime numbers (<=8 digits) as the process of generating and checking prime factors at the same time would increase the time complexity and decrease the overall efficiency of the program. The prime numbers are fetched from the list and the modulus is calculated with public key (large input value) till a positive result is found. Since the public key is a product of 2 large prime numbers, on finding one the other is found easily. The whole process is carried out using the following in OpenMP :-

1. Nested Parallelisation
2. Dynamic scheduling

Along with the above mentioned, “*num\_threads()*” and “*private*” clauses were also used.

The difference between parallel and serial execution times were found out to be minimal for a public key of less than or equal to 10 digits. It was for only larger numbers that parallel execution time was ***considerably less*** than the serial execution time. Therefore, RSA cracking was **successfully optimised** using *parallelisation*.

**GRAPHS** –

X-axis : NUMBER OF THREADS

Y-axis : TIME ( in ms scientific notation )