Parallelized RSA Algorithm: An Analysis With Performance Evaluation using OpenMP Library in High Performance Computing Environment

Md. Ahsan Ayub, Zishan Ahmed Onik, Steven Smith

Department of Computer Science

Tennessee Technological University

{mayub42, zonik42, smsmith23}@students.tntech.edu

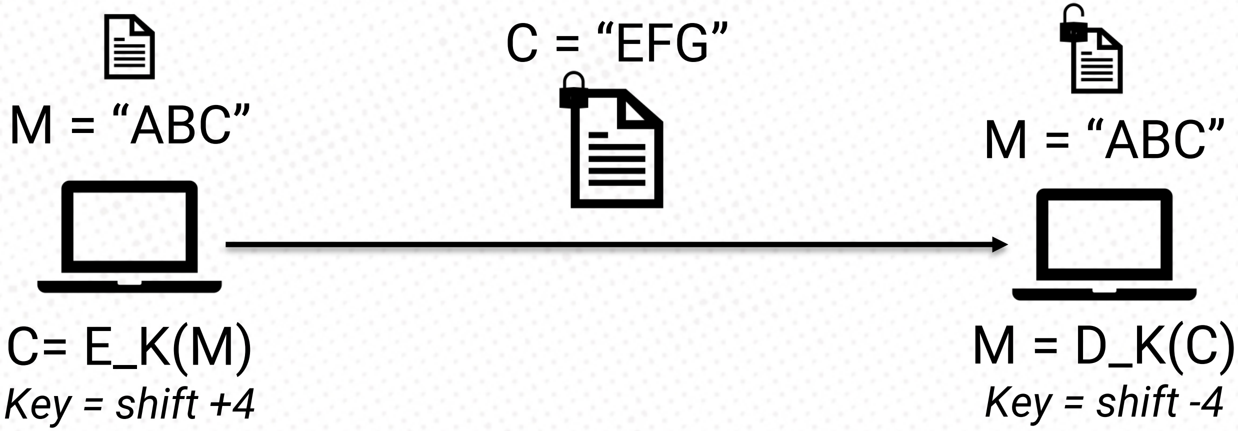
CEROC
Cybersecurity Education, Research and Outreach Center

December 19, 2019

Content

- Introduction
- Methodology
- Results
- Conclusion

Basic Encryption



Symmetric Encryption

- The encryption / decryption keys are the same (K)
- An example of symmetric key encryption based communication –
  - Alice and Bob agree on a cryptosystem (e.g. AES)
  - Alice and Bob agree on a (secret) key K
  - Alice encrypts a message using K and she sends it to Bob $C = E\_K(M)$
  - Bob decrypts the message using K, $M = D\_K(C)$
- Faster implementation

Asymmetric Encryption

- There are two different keys (K1, K2)
- It is not possible (or computationally feasible) to calculate K1 given K2 or vice versa
- Encryption with one key, decryption with the other key
- An example –
  - Alice and Bob agree on a public-key crypto system (e.g., RSA)
  - Bob's public key PKb is sent to Alice
  - Alice encrypts a message with Bob's public key and sends it
  - Bob decrypts the message using his private key SKb
- Slower implementation

Goals: Asymmetric Encryption

- The process of encrypting and decrypting a message will be computationally feasible.
- Deriving the private key from the public key of a particular user should be computationally infeasible.
- Deriving the private key from a chosen plaintext should also be computationally infeasible.

Methodology

RSA Algorithm

Algorithm 1 Mathematical Operations in the RSA Algorithm

Input: p, q

Output: n, e, d

1) Choose two large prime numbers, p and q
2) Compute: $n = p \times q$
3) Compute: Euler totient function [29], $\phi(n) = (p-1) \times (q-1)$
4) Choose $1 < e < n$ such that e is relatively prime to $\phi(n)$
5) Compute: $d = e^{-1} \bmod \phi(n)$

A Use Case of RSA Algorithm

p
121310724392112718973236715316124404284724276337014109256
345493123019643734208561932419736532241686654101705736136
52141717117137979742993348710628298 03541

q
120275242554787488859562207937345121287333878036820754336
538999839551798509887978998691469008091316111533468170508
320960221601463663463918124709871054152 33

n (derived from $n = p \times q$)
145906768007583323230186939349070635292401872375357164399
581871019873438799005358938369571402670149802121818086292
467422828157022922076746906543401224889672472407926969987
100581290103199317858753663710862357656510507883714297115
633427889114635351027120327651665184117268598379886721118
37205085526346618740053

$\boldsymbol{\phi(n)}$ (derived from $\phi(n) = (p-1) \times (q-1)$)
145906768007583323230186939349070635292401872375357164399
581871019873438799005358938369571402670149802121818086292
467422828157022922076746906543401224889648313811232279966
317301397777852365301547848273478871297222058587457152891
606459269718119268971163555070802643999529549644116811947
516513938184296683521280

e (selected from $1 < e < n$ such that e is relatively prime to $\phi(n)$)
65537

d (derived from $d = e^{-1} \bmod \phi(n)$)
894894250092744443682285459217730939196695860658842574454
978544564876748396298183909349419732628796167979706089172
836798754993315741611138540888132754881105882471930775825
272784379065040156806234235500672400424666656542323835029
222154936232894721388664458187891279461234078077257026266
44091036502372545139713

Our Major Contribution

- Due to the benefits of the RSA algorithm, the necessity of faster implementations will continue to grow.
- In our study, we parallelize the exponentiation operations of the RSA algorithm in a high performance computing environment.

$$C = E(M) = (M^{d_{\text{(sender)}}} \mod n)^{e_{\text{(receiver)}}} \mod n$$

$$M = D(C) = (C^{d_{\text{(receiver)}}} \mod n)^{e_{\text{(sender)}}} \mod n$$

Results

Experimental Parameters

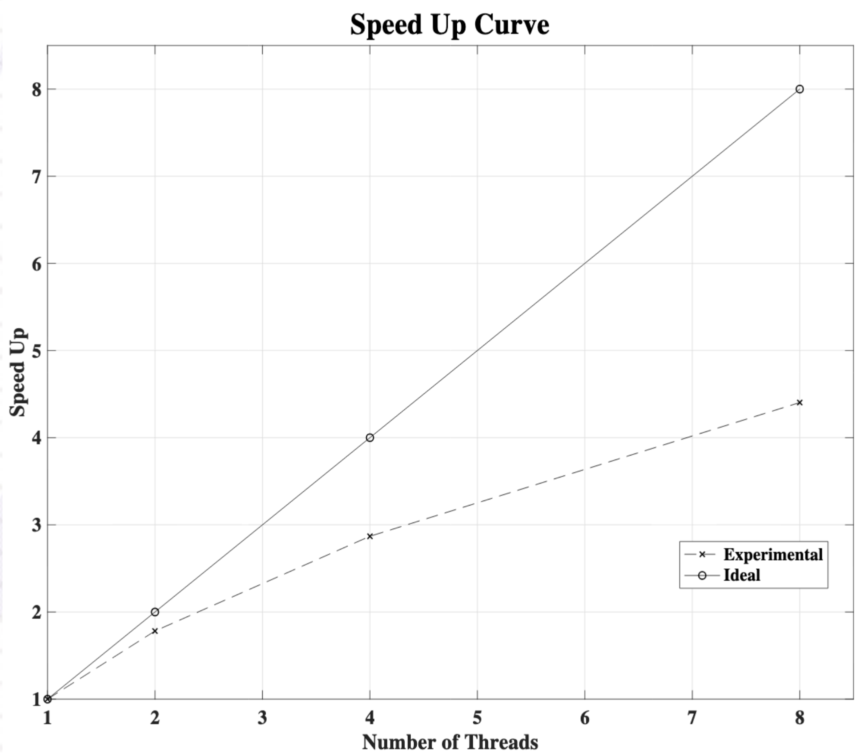
$$Speed\ Up,\ S = \frac{T_S}{T_P}$$

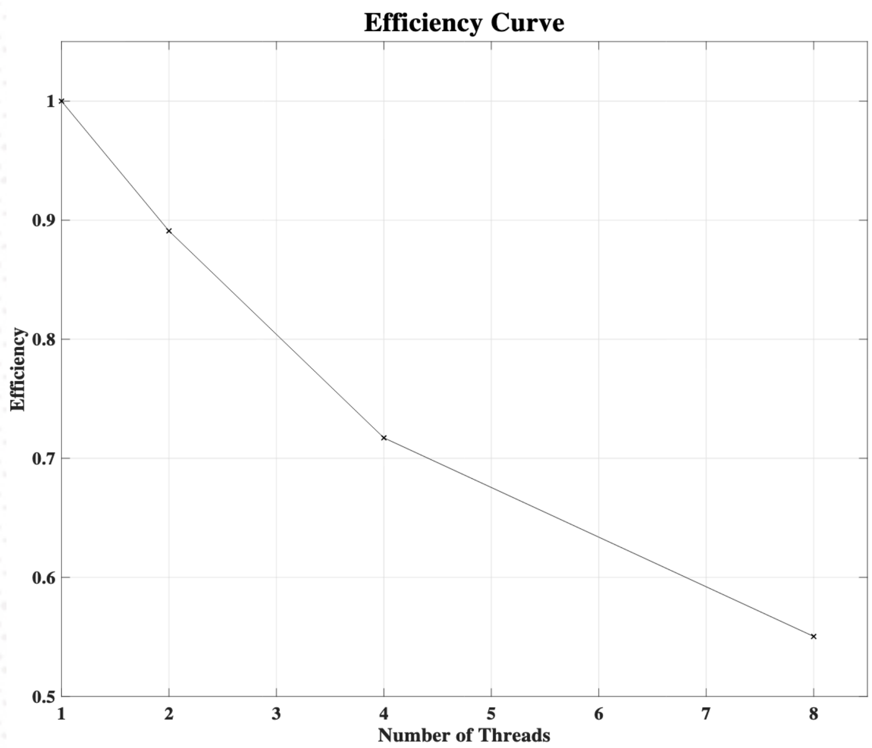
$$Efficiency,\ E = \frac{S}{p}$$

Experimental Findings

Threads	1^{st} Run Time	2^{nd} Run Time	3^{rd} Run Time	Avg. Run Time	Speed Up	Efficiency
1	4.5	4.7	4.6	4.6	1.00	1.00
2	2.7	2.6	2.5	2.6	1.782	0.891
4	1.7	1.5	1.6	1.6	2.8688	0.7172
8	1.0	1.1	0.9	1.0	4.4033	0.5504

Experimental Findings (Cont.)





Conclusion

Summary

- Different avenues of the RSA parallelization techniques discussed in our paper (a survey of 1978 till date)
- Parallelize exponentiation operation of the RSA algorithm using OpenMP library in HPC environment
- RSA is not a classic parallelizable problem, e.g., Matrix Multiplication
- In the spirit of open science, our developed tool has made open source and available online at –

`https://github.com/AhsanAyub/RSAParallelization`

Future Work

- Incorporating the Chinese Remainder Theorem (CRT) to optimize modular operations
- Utilizing another high performance message passing computing library, such as, Message Passing Interface (MPI), and then provide a comparison
- Focusing on GPU-based implementation

Acknowledgement

- Anonymous reviewers
- Dr. Ambareen Siraj and Dr. Sheikh Ghafoor
  - Professor, Dept. of Computer Science, Tennessee Tech University
- Cybersecurity Education, Research and Outreach Center (CEROC) at Tennessee Tech University

THANK YOU!

Happy to take any questions you may have!