ENHANCEMENT OF ENCRYPTION ALGORITHMS BY PARALLELIZING

SAURAB KHANAL - 17BCE2345
SHAIKAT DAS JOY - 17BCI0198
ANTON MARCUS - 17BCE0045
SANCHET KUMAR SAHU-17BCE0595
G.B.S. SATHVIK - 18BCI0087

ABSTRACT

Encryption Algorithms are necessary part of the data sharing today as every bit of data need encryption so that it could be transferred from one place to another without the fear of data leaking. Every lost bit of data puts the system to danger. Encryption algorithms compose of necessary steps involving a key that helps converting plain text to cipher text. All the steps must be performed carefully to ensure that the data encrypted is found back. Thus, many algorithms are made to run sequentially and in serial. This makes the algorithm to take much computation time.

We aim to reduce the total computation time needed to complete the process by parallelizing the algorithm using the concept of multi-threading. The complexity, data structures and the overall procedure of the algorithm would remain constant.

PROBLEM DEFINITION

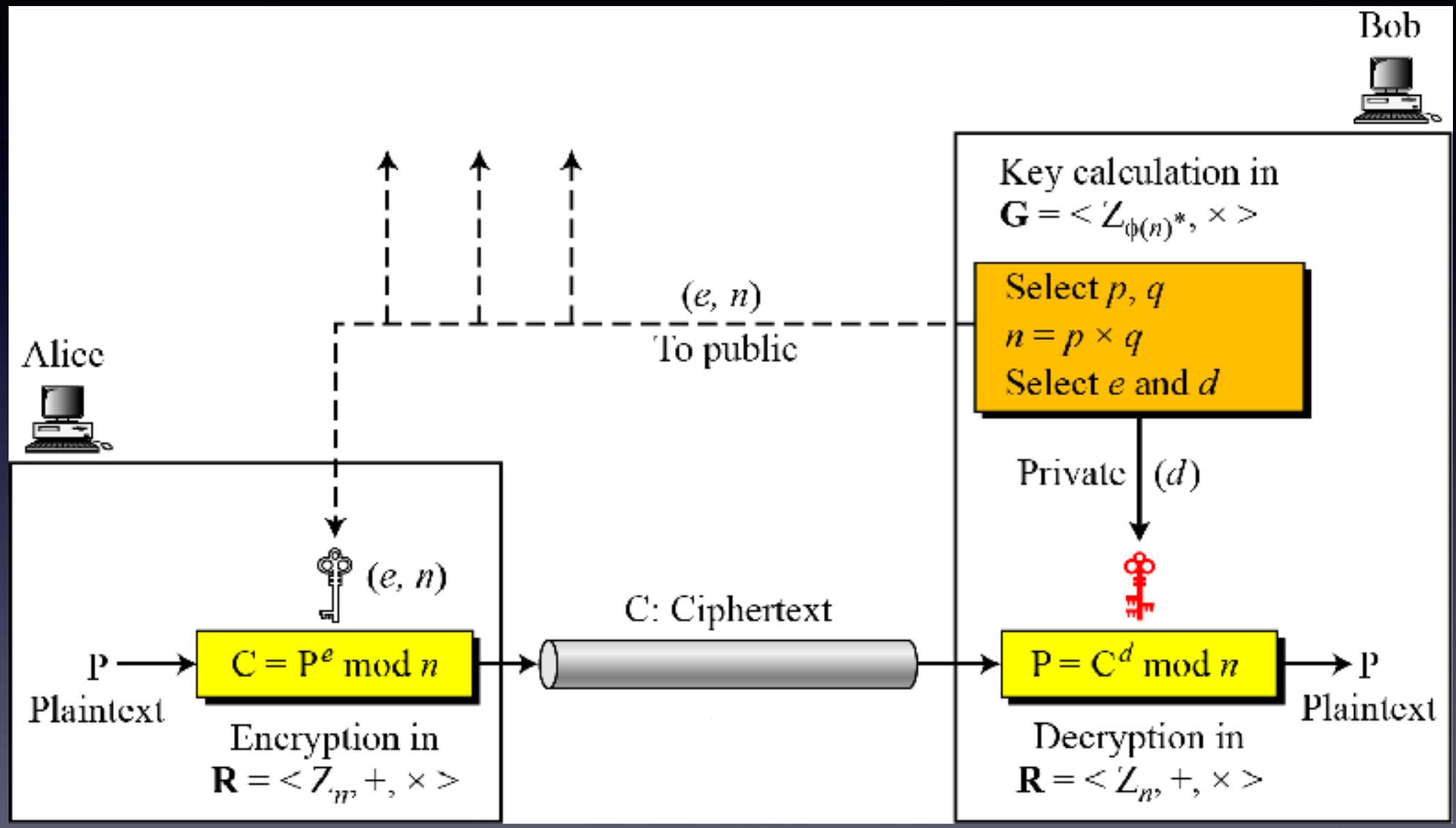
Encryption Algorithm comprises of a number of steps that are generally performed sequentially. Our problem focuses on parallelizing these algorithms. The algorithms are analysed for ways to break it down to remove all the data dependencies and inter-dependent steps that can be resolved so as to be executed by different threads in a parallel manner.

LITERARY SURVEY

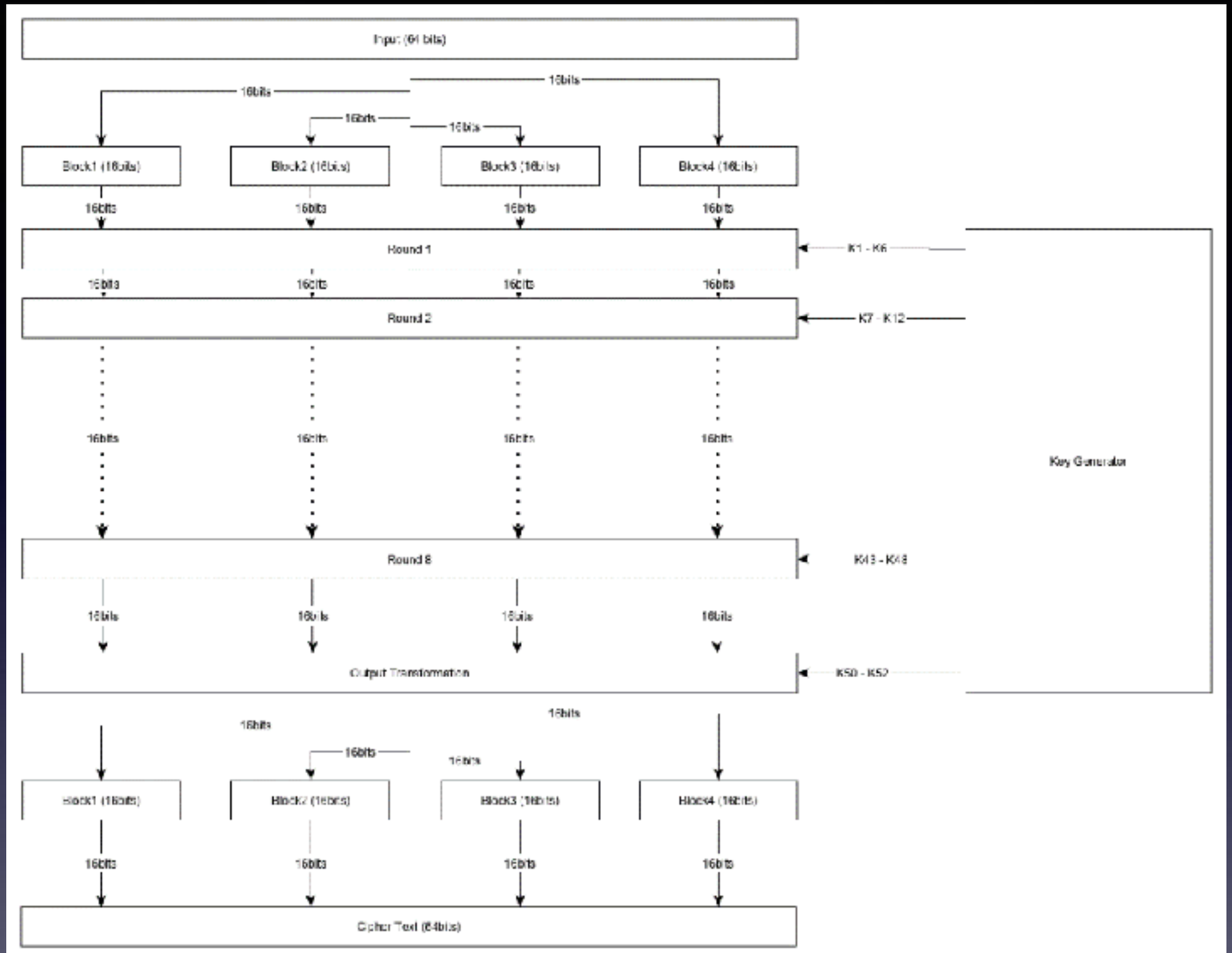
Authors	**Journal**	**Title**	**Methodolgy**	**Merits**	**Demerits**
A. Joseph Amalraj1 , Dr. J. John Raybin Jose	International Journal of Computer Science and Mobile Computing	A SURVEY PAPER ON CRYPTOGRAPHY TECHNIQUES	Using public key infrastructure to check email security	Perspective and the procedure of cryptography assumes a significant job to give the security to the system	Innovation, there are such huge numbers of things that offers office to manage these innovation utilizing web.
Dr. Prerna Mahajan & Abhishek Sachdeva	Global journal of computer science and technology	A Study of Encryption Algorithms AES, DES and RSA for Security	AES, DES and RSA algorithms and compared their performance of encrypt techniques based on the analysis	Encryption calculation assumes significant job in correspondence security.	It tooks more time and for big project works slowly
Venkat Prasad.K, S. Magesh	International Journal of Pure and Applied Mathematics	A SURVEY ON ENCRYPTION ALGORITHMS USING MODERN TECHNIQUES	survey of DES, RSA algorithms and their performance analysis	ensures secure communication and data security ensures data confidentiality, authentication.	Expensive and taking more time to work i nsome cases not work properly
Yahia Alemami, Mohamad Afendee Mohamed, Saleh Atiewi	International Journal of Recent Technology and Engineering (IJRTE)	Research on Various Cryptography Techniques	preserving information privacy and secrecy using ASCII algorithms	methodology is effective, fast, secure and reliable.	AES algorithm and Diffie–Hellman is proposed before sending data to the cloud.

Ching-Chao Yang, Tian-Sheuan Chang, and Ch	IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS—II: ANALOG AND DIGITAL SIGNAL	A New RSA Cryptosystem Hardware Design Based on Montgomery	Montgomery's modular multiplication algorithm	algorithm efficiently reduces the critical can, avoid the final adjustment of residue, save the data format conversion time.	No undo option for modification. Execution time is very high. Chance of cyber attack at any time of execution.
Wei Wang, Xinming Huang,	IEEE Transactions on Circuits and Systems II: Express Briefs	A Novel and Efficient Design for RSA Cryptosystem with Very Large Key Size	Montgomery multiplier is synthesized for ASIC implementation using Synopsys Design Compiler.	48K-bit RSA design outperforms others with respect to throughput and efficiency.	architecture is very complicated
c.-c. Chang and M.-S. Hwang	ELECTRONICS LETTERS 18th July 1996 Vol. 32 No. 15	Parallel computation of the generating keys for RSA cryptosystems	parallel implementation for generating RSA keys without the Euclidean algorithm.	efficient approach to assist an end user to choose a public key e	errors while execution theoretical implementation
Rami Aldahdooh		Parallel Implementation and Analysis of Encryption Algorithms	execution of the cryptographic calculations with a specific end goal to accelerate the encryption procedure	scalability is much easier, the time of processing can be decreases when the keys are longer	not for normal computer users scalability limitation of the dedicated hardware.
Vishal Pachori, Gunjan Ansari, Neha Chaudhary	International Journal of Engineering Research and Applications (IJERA)	Improved Performance of Advance Encryption Standard using Parallel Computing	data parallelism and control parallelism	operations are done on server side so that there is no need to send data to other processing unit	The network traffic can affect the performance of this implementation.
Naser Attar, Hossein Deldari, Marzie Kalantari	Canadian Center of Science and Education	AES Encryption Algorithm Parallelization in Order to Use Big Data Cloud	128-bit AES is used which divides the data into 4 equal pieces and 4 processor perform the calculation in parallel.	keys are trasnferred physically or divided into several part to	Storing and processing of the data in a place other than their own organization, is not acceptable by users.

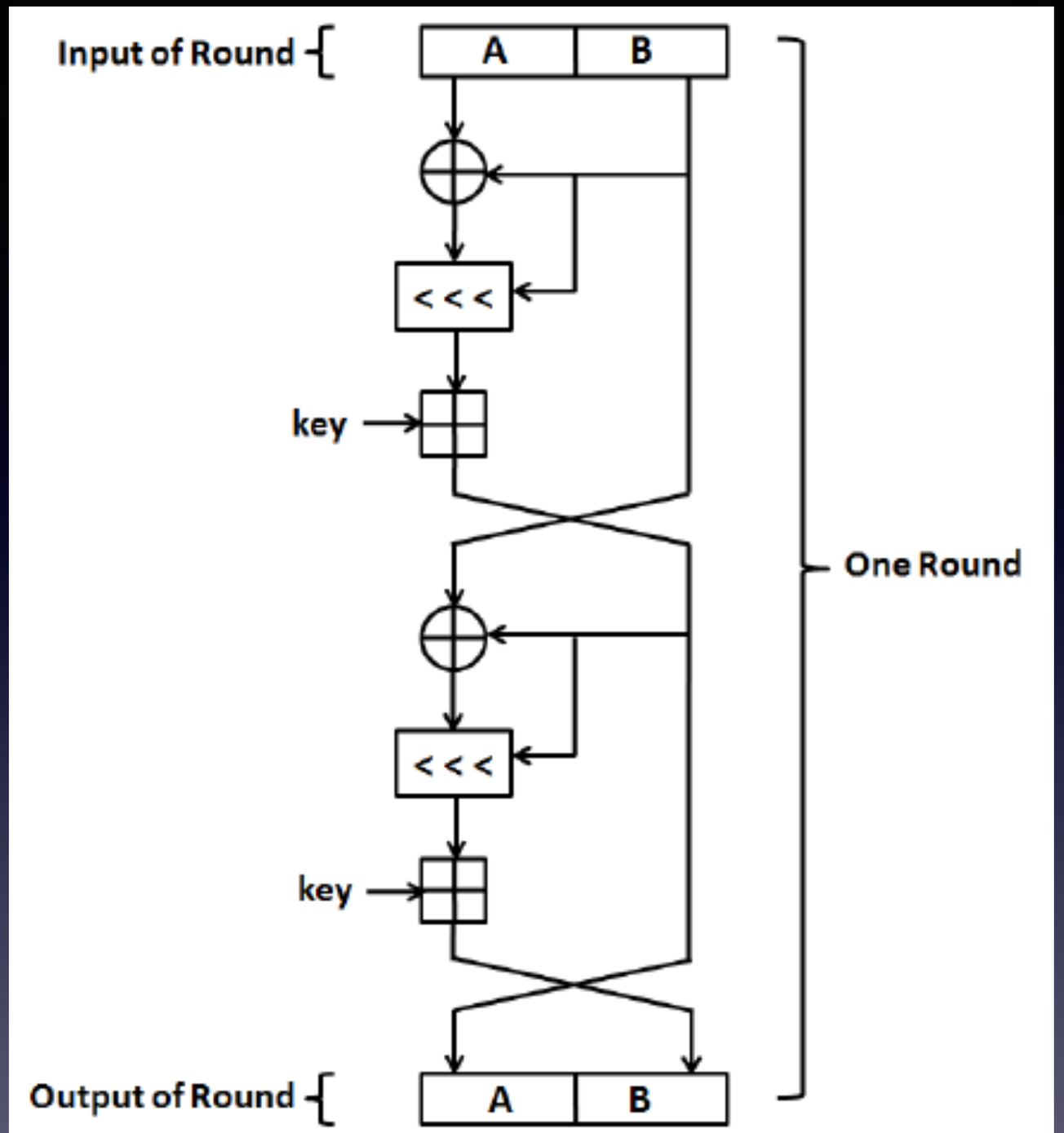
BLOCK DIAGRAMS



PROCEDURE OF RSA ALGORITHM

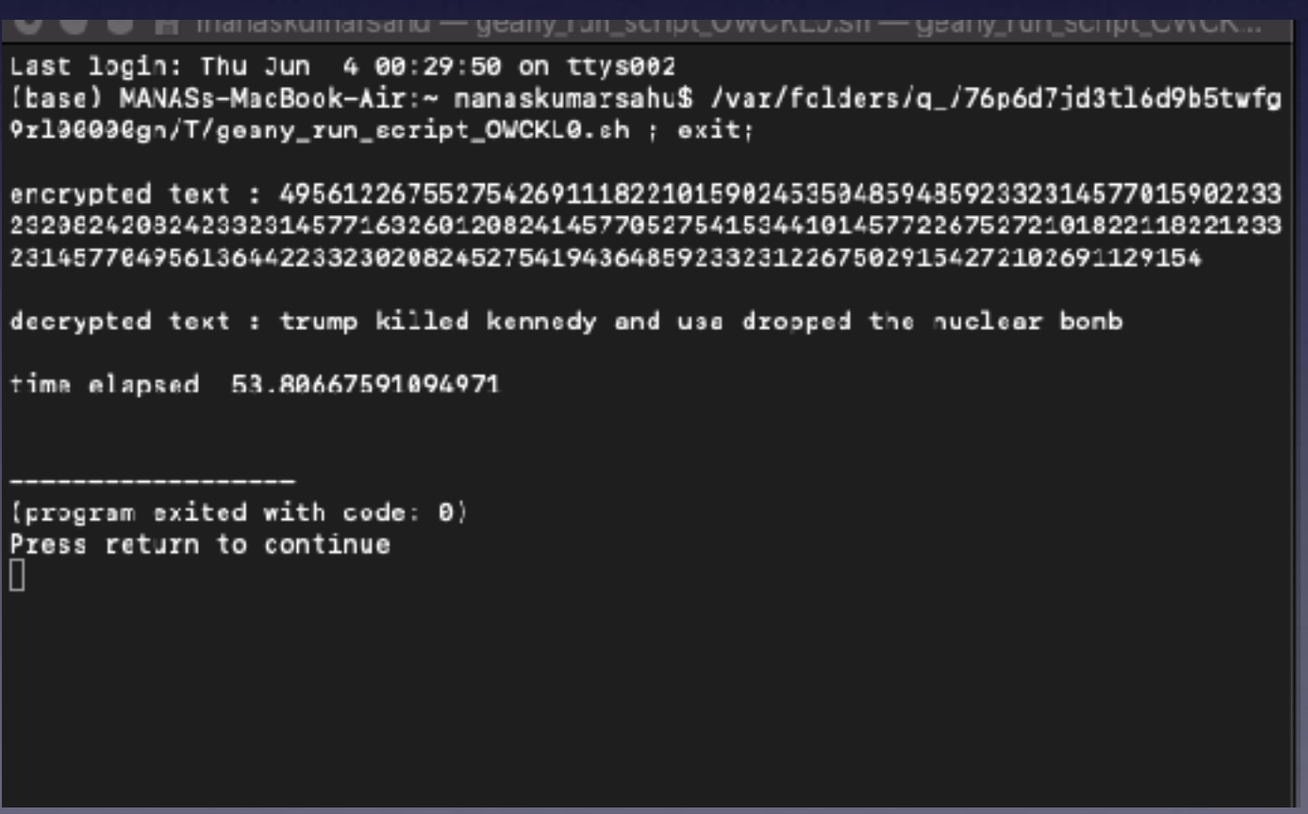


Block diagram for IDEA Algorithm

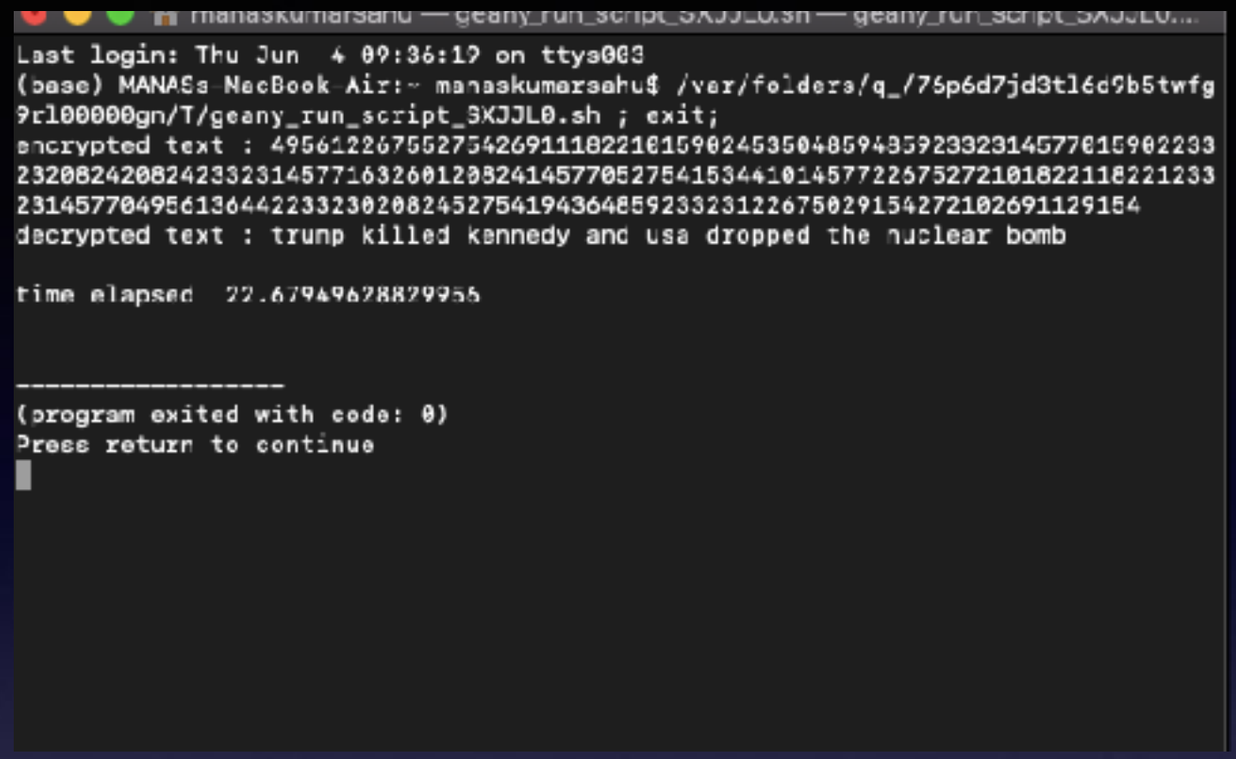


Block diagram for the one-round in RC5

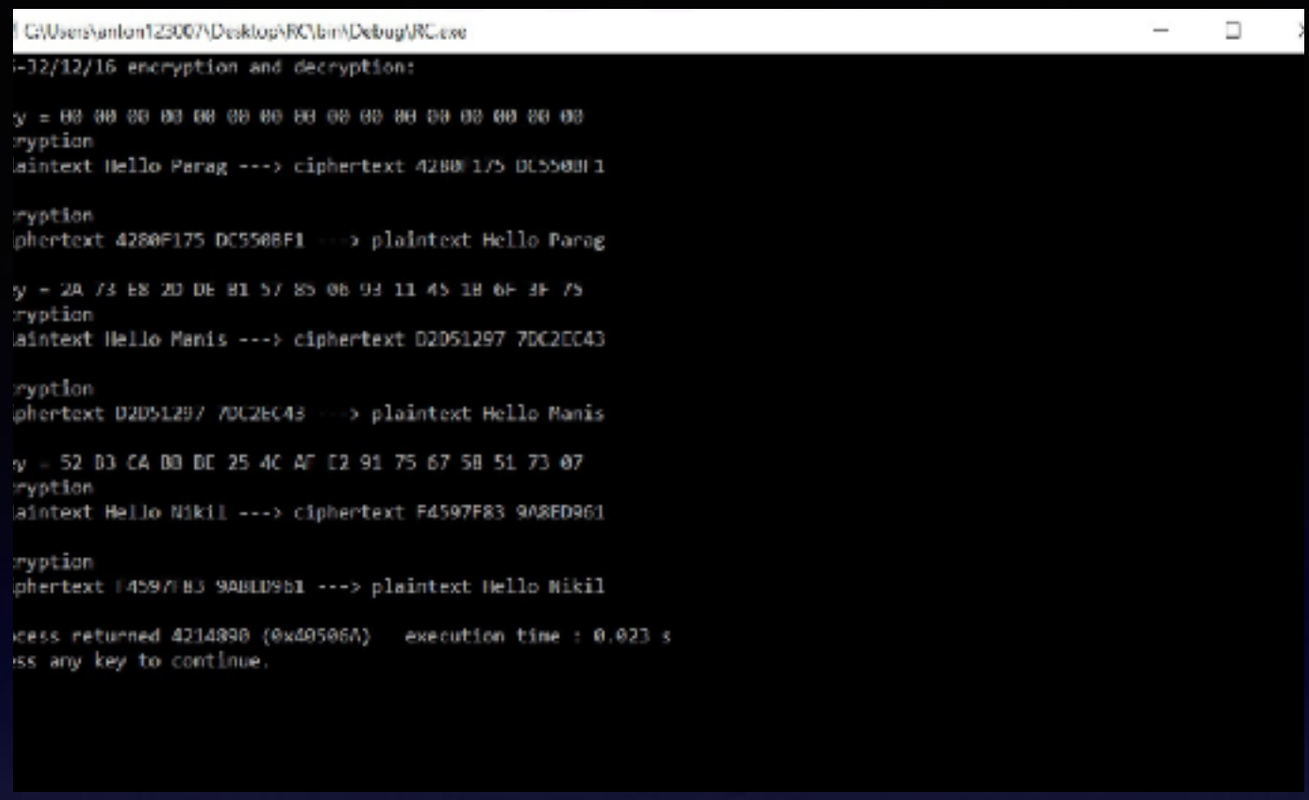
IMPLEMENTATION



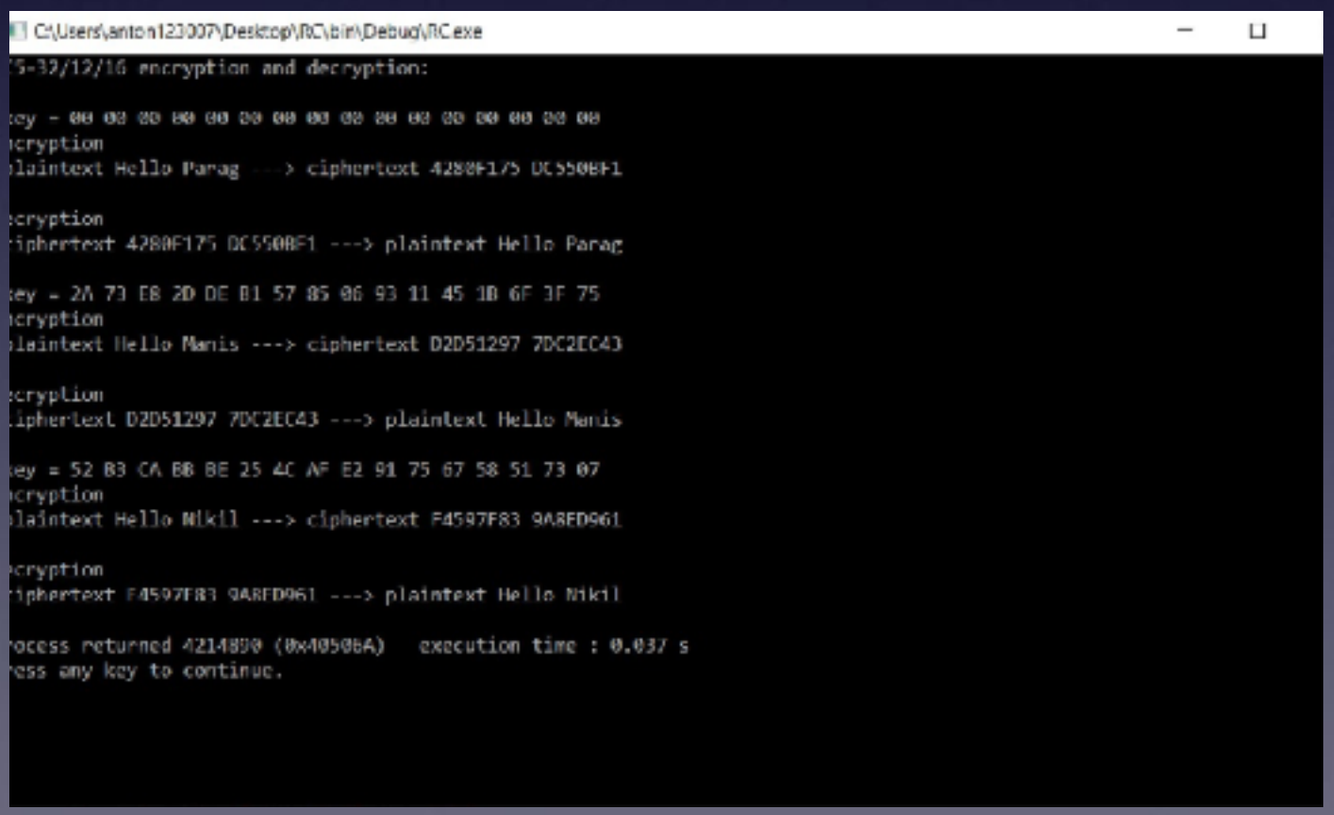
Serial Execution
Time taken = 53.807



Parallel Execution
Time taken = 22.679

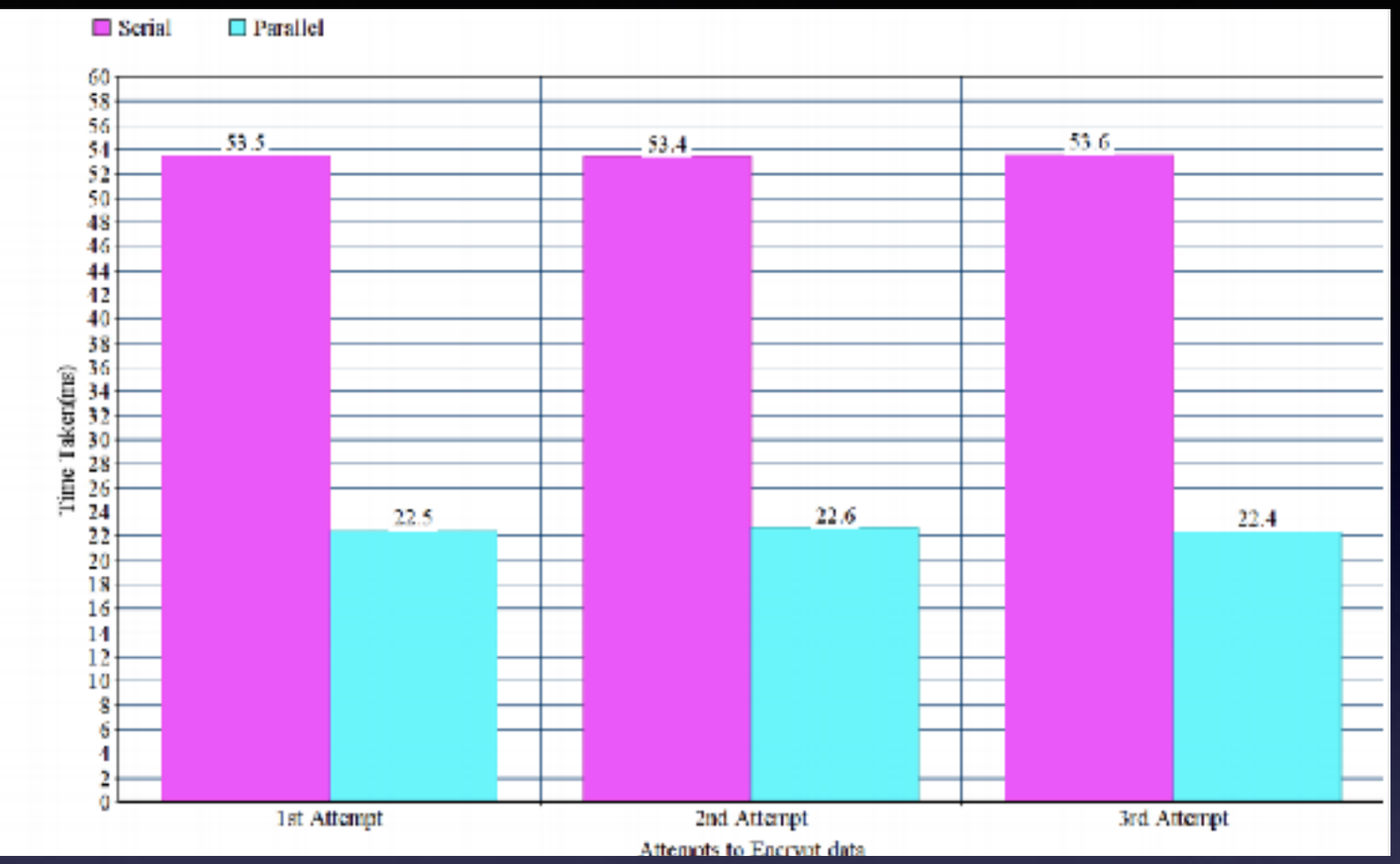


Parallel Execution
Time taken = 0.023s



Serial Execution
Time taken = 0.037s

GRAPHICAL COMPARISON



RSA Algorithm

RC5 Algorithm

CONCLUSION

Advantages of parallelization :

- Efficient code
- Less execution times
- Distribution of instructions to threads automated

- **Disadvantages of Parallelization :**
- Overhead cost outweighs the time saved in case of small strings
- Memory usage multiplies by the number of threads/processors used
- Weight of performance on processor increases exponentially with the increase in number of processors

REFERENCES

- Amalraj, A. J., & Jose, J. J. R. (2016). A survey paper on cryptography techniques. *International Journal of Computer Science and Mobile Computing*, *5*(8), 55-59.
- Mahajan, P., & Sachdeva, A. (2013). A study of encryption algorithms AES, DES and RSA for security. *Global Journal of Computer Science and Technology*.
- Venkat Prasad K., & Magesh S. (2015). A SURVEY ON ENCRYPTION ALGORITHMS USING MODERN TECHNIQUES. International Journal of Pure and Applied Mathematics
- Alemami, Y., Mohamed, M. A., & Atiewi, S. Research on Various Cryptography Techniques.
- Yang, C. C., Chang, T. S., & Jen, C. W. (1998). A new RSA cryptosystem hardware design based on Montgomery's algorithm. *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, *45*(7), 908-913.
- Huang, X., & Wang, W. (2015). A novel and efficient design for an RSA cryptosystem with a very large key size. *IEEE Transactions on Circuits and Systems II: Express Briefs*, *62*(10), 972-976.
- Chang, C. C., & Hwang, M. S. (1996). Parallel computation of the generating keys for RSA cryptosystems. *Electronics Letters*, *32*(15), 1365-1366.
- Rami Aldahdooh. Parallel Implementation and Analysis of Encryption Algorithms
- Pachori, V., Ansari, G., & Chaudhary, N. (2012). Improved performance of advance encryption standard using parallel computing. *International Journal of Engineering Research and Applications (IJERA)*, *2*(1), 967-971.
- ATTAR, N., DELDARI, H., & KALANTARI, M. (2017). AES ENCRYPTION ALGORITHM PARALLELIZATION IN ORDER TO USE BIG DATA CLOUD.

THANK YOU!!!

ENHANCING ENCRYPTION ALGORITHMS THROUGH PARALLELIZING

A PROJECT REPORT

Submitted by

Saurab Khanal (17BCE2345)
Shaikat Das Joy (17BCI0198)
Anton Marcus (17BCE0045)
Sanchet Kumar Sahu (17BCE0595)
G.B.S. Sathvik (18BCI0087)

CSE4003 – CYBER SECURITY

Under the guidance of
Prof. Manikandan K.,
SCOPE, VIT , Vellore.

VIT®
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

June, 2020

Table of Contents

1. Abstract

Encryption Algorithms are necessary part of the data sharing today as every bit of data need encryption so that it could be transferred from one place to another without the fear of data leaking. Every lost bit of data puts the system to danger. Encryption algorithms compose of necessary steps involving a key that helps converting plain text to cipher text. All the steps must be performed carefully to ensure that the data encrypted is found back. Thus, many algorithms are made to run sequentially and in serial. This makes the algorithm to take much computation time.

We aim to reduce the total computation time needed to complete the process by parallelizing the algorithm using the concept of multi-threading. The complexity, data structures and the overall procedure of the algorithm would remain constant.

Keywords: Parallelize, encryption algorithms, multi-threading.

2. Introduction

2.1. IDEA

The Data Encryption Standard (DES) algorithm has been a popular secret key encryption algorithm and is used in many commercial and financial applications.

Although introduced in 1976, it has proved resistant to all forms of cryptanalysis. However, its key size is too small by current standards and its entire 56-bit key space can be searched in approximately 22 hours.

International Data Encryption Algorithm (IDEA) is a block cipher designed by Xuejia Lai and James L. Massey of ETH-Zürich and was first described in 1991. It is a minor revision of an earlier cipher, PES (Proposed Encryption Standard); IDEA was originally called IPES (Improved PES). IDEA was used as the symmetric cipher in early versions of the Pretty Good Privacy cryptosystem.

IDEA was to develop a strong encryption algorithm, which would replace the DES procedure developed in the U.S.A. in the seventies. It is also interesting in that it entirely avoids the use of any lookup tables or S-boxes. When the famous PGP email and file encryption product was designed by Phil Zimmermann, the developers were looking for maximum security. IDEA was their first choice for data encryption based on its proven design and its great reputation.

The IDEA encryption algorithm provides high level security not based on keeping the algorithm a secret, but rather upon ignorance of the secret keyis fully specified and easily understood is available to everybody is suitable for use in a wide range of applications can be economically implemented in electronic components (VLSI Chip) can be used efficiently may be exported world wide is patent protected to prevent fraud and piracy

2.2. RSA

RSA algorithm, invented by Rivest, Shamir and Adelman in 1978, is one of the famous algorithms for public-key cryptography. It is appropriate for encryption and digital signature. RSA is the utmost far used algorithm in Internet security . In fact, Internet security depends significantly on the security properties of the RSA cryptosystem. Its security depends upon the insolvability of the integer factorization problem and is believed to be vulnerable given sufficiently long keys, such as 1024 bits or more.

To guard data transmitted from snooping by someone other than the receiver. It is desired to hide the

message before it is sent to a non-secure communication channel. This is achieved through encryption. Due to its distinctive ability to distribute and manage keys, public key encryption has become the perfect solution to information security . Public key algorithms (e.g., RSA algorithm) rely essentially on hard mathematical problems (modular multiplication and modular exponentiation) of very large integers, ranging from them.

2.3. Rivest Cipher 5

RC5 is a should be a symmetric block cipher. The same secret cryptographic key is used for encryption and for decryption. The plaintext and ciphertext are fixed-length bit sequences (blocks). RC5 is suitable for hardware or software. This means that RC5 is used only computational primitive operations commonly found on typical microprocessors. RC5 is fast. This more-or-less implies that RC5 is word-oriented: the basic computational operations should be operators that work on full words of data at a time.

RC5 is adaptable to processors of different word-lengths. For example, as 64-bit processors become available, it should be possible for RC5 to exploit their longer word length. Therefore, the number w of bits in a word is a parameter of RC5; different choices of this parameter result in different RC5 algorithms. RC5 should be iterative in structure, with a variable number of rounds. The user can explicitly manipulate the trade-off between higher speed and higher security. The number of rounds r is a second parameter of RC5. RC5 should have a variable-length cryptographic key. The user can choose the level of security appropriate for his application, or as required by external considerations such as export restrictions. The key length b (in bytes) is thus a third parameter of RC5.

RC5 is simple. It is easy to implement. More importantly, a simpler structure is perhaps more interesting to analyse and evaluate, so that the cryptographic strength of RC5 can be more rapidly determined. RC5 has a low memory requirement, so that it may be easily implemented on smart cards or other devices with restricted memory. RC5 provides high security when suitable parameter values are chosen. In addition, during the development of RC5, we began to focus our attention on an intriguing new cryptographic primitive: data-dependent rotations, in which one word of intermediate results is cyclically rotated by an amount determined by the low-order bits of another intermediate result. RC5 highlights the use of data-dependent rotations, and encourage the assessment of the cryptographic strength data-dependent rotations can provide.

3. Literature Review

Authors	Journal	Title	Methodolgy	Merits	Demerits
A. Joseph Amalraj1 , Dr. J. John Raybin Jose	International Journal of Computer Science and Mobile Computing	A SURVEY PAPER ON CRYPTOGRAPHY TECHNIQUES	Using public key infrastructure to check email security	Perspective and the procedure of cryptography assumes a significant job to give the security to the system	Innovation, there are such huge numbers of things that offers office to manage these innovation utilizing web.
Dr. Prerna Mahajan & Abhishek Sachdeva	Global journal of computer science and technology	A Study of Encryption Algorithms AES, DES and RSA for Security	AES, DES and RSA algorithms and compared their performance of encrypt techniques based on the analysis of its stimulated time at the time of encryption and decryption	Encryption calculation assumes significant job in correspondence security.	It tooks more time and for big project works slowly
Venkat Prasad.K, S. Magesh	International Journal of Pure and Applied Mathematics	A SURVEY ON ENCRYPTION ALGORITHMS USING MODERN TECHNIQUES	survey of DES, RSA algorithms and their performance analysis	ensures secure communication and data security ensures data confidentiality, authentication, integrity, availability and identification	Expensive and taking more time to work i nsome cases not work properly
Yahia Alemami, Mohamad Afendee Mohamed, Saleh Atiewi	International Journal of Recent Technology and Engineering (IJRTE)	Research on Various Cryptography Techniques	preserving information privacy and secrecy using ASCII algorithms	methodology is effective, fast, secure and reliable.	AES algorithm and Diffie–Hellman is proposed before sending data to the cloud.
Ching-Chao Yang, Tian-Sheuan Chang, and Ch	IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS—II: ANALOG	A New RSA Cryptosystem Hardware Design Based on Montgomery	Montgomery's modular multiplication algorithm	algorithm efficiently reduces the critical can, avoid the final adjustment of residue, save	No undo option for modification. Execution time is very high. Chance of cyber attack at

	AND DIGITAL SIGNAL PROCESSING, VOL. 45, NO. 7, JULY 1998			the data format conversion time.	any time of execution.
Wei Wang, Xinming Huang,	IEEE Transactions on Circuits and Systems II: Express Briefs	A Novel and Efficient Design for RSA Cryptosystem with Very Large Key Size	Montgomery multiplier is synthesized for ASIC implementation using Synopsys Design Compiler, DesignWare	48K-bit RSA design outperforms others with respect to throughput and efficiency.	architecture is very complicated
c.-c. Chang and M.-S. Hwang	ELECTRONIC S LETTERS 18th July 1996 Vol. 32 No. 15	Parallel computation of the generating keys for RSA cryptosystems	parallel implementation for generating RSA keys without the Euclidean algorithm.	efficient approach to assist an end user to choose a public key e	errors while execution theoretical implementation
Rami Aldahdooh	*International* Workshop on Cryptographic Hardware and Embedded Systems	Parallel Implementation and Analysis of Encryption Algorithms	execution of the cryptographic calculations with a specific end goal to accelerate the encryption procedure and utilize the new multi-core processors productively.	scalability is much easier, the time of processing can be decreases when the keys are longer	not for normal computer users scalability limitation of the dedicated hardware.
Vishal Pachori, Gunjan Ansari, Neha Chaudhary	International Journal of Engineering Research and Applications (IJERA)	Improved Performance of Advance Encryption Standard using Parallel Computing	data parallelism and control parallelism	operations are done on server side so that there is no need to send data to other processing unit	The network traffic can affect the performance of this implementation .
Naser Attar, Hossein Deldari, Marzie Kalantari	Canadian Center of Science and Education	AES Encryption Algorithm Parallelization in Order to Use Big Data Cloud	128-bit AES is used which divides the data into 4 equal pieces and 4 processor perform the calculation in parallel. As the	keys are trasnferred physically or divided into several part to	Storing and processing of the data in a place other than their own organization, is not acceptable by users.

			environment is inherently distributed with high data volume		

4. Problem Description

Encryption Algorithm comprises of a number of steps that are generally performed sequentially. Our problem focuses on parallelizing these algorithms. The algorithms are analysed for ways to break it down to remove all the data dependencies and inter-dependent steps that can be resolved so as to be executed by different threads in a parallel manner.

Hence two cases can be made from the above scenario

1. The algorithm steps as well as the pre-processing can be parallelized along with the instruction
2. The algorithm steps cannot be parallelized but pre-processing can be parallelized along with the instruction.
3. Neither the algorithm steps nor the pre-processing can be parallelized but instructions can be parallelized

5. Hardware & Software Requirements

To run our code in a system, it should complete the following requirements

4. Java SE 8 update 201 and above, Java SE 10, 11 and 12
5. Python 3.x
6. GCC 7.4 and above

The processor can be multiple core or a single core. The processor should not be single threaded. The performance of all the programs have been tested on Intel i5 8th Generation processors.

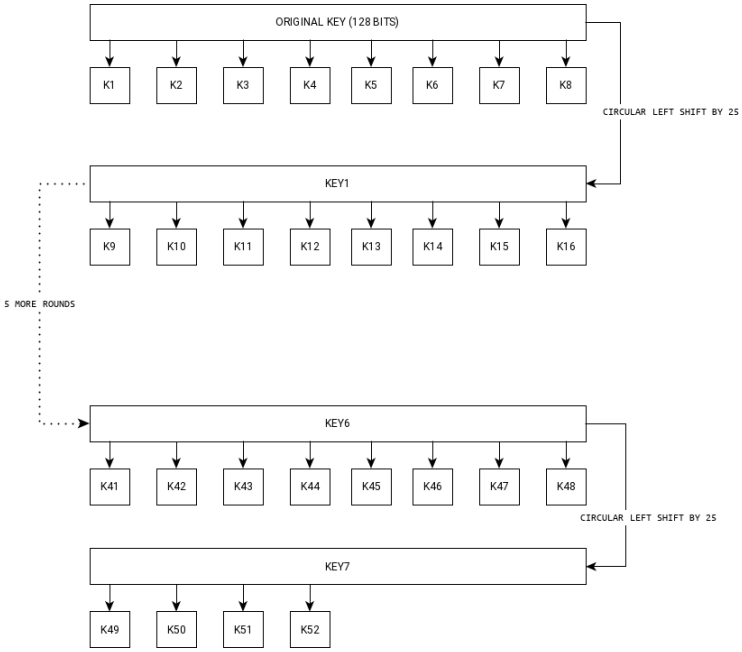
6. System Design

6.1 IDEA

IDEA is a block cipher. It operates on blocks of text. According to IDEA standards, the input text for the algorithm is 64bits in length. The algorithm divides the input text into blocks of 16bits each and operates them with sub-keys of length 16bits. The key is 128bits in length. There is a total of 52 sub-keys. The first 8 rounds use 6 sub-keys while the last round use 4.

Key Generation Process

The key is 128bits in length. This key is used to generate 8 sub-keys. The first 6 being sub-keys of the first round while the 2 keys be the first 2 sub-keys of the second round. The original key is then shifted left (circularly) by 25 and the resulting key is used to generate 8 other sub-keys.



Decryption Key Generation:

For generating keys for decryption, the following equation is followed:

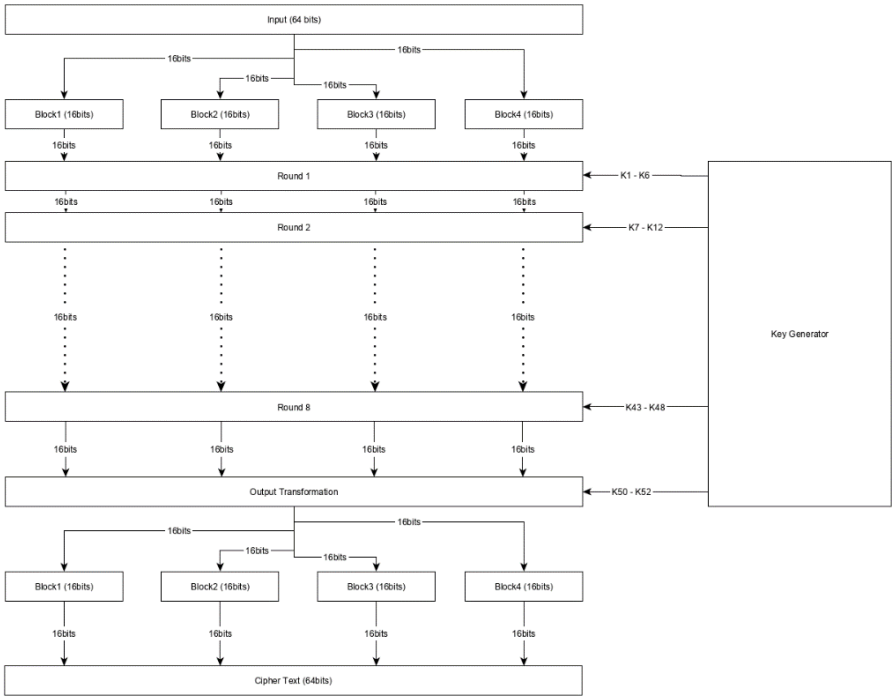
$$?_1 = \frac{1}{K_{49}}$$
$$K_2 = -K_{50}$$
$$K_3 = -K_{51}$$
$$?_4 = \frac{1}{K_{52}}$$
$$K_5 = K_{47}$$
$$K_5 = K_{48}$$

Here, $1 / K_i$ indicates the multiplicative inverse for K_i and $-K_i$ indicates the 2's compliment of K_i

Algorithm

There are 9 rounds in IDEA. The first 8 are called as general rounds and they use 6 16bits sub-keys each. The operations performed in the general rounds are

7. Multiplication Modulo $2^{16}+1$
8. Addition Modulo $2^{16}+1$
9. Xclusive OR



The general round is performed as:

STEP 1 : Multiply PT1 and Key1
STEP 2 : Add PT2 and Key 2
STEP 3 : Add PT3 and Key 3
STEP 4 : Multiply PT4 and Key K4
STEP 5 : Result of Step 1 XOR result of step 3
STEP 6 : Result of Step 2 XOR result of step 4
STEP 7 : Multiply step 5 with Key 5
STEP 8 : Add result of step 6 and step 7
STEP 9 : Multiply result of step 8 with Key 6.
STEP 10 : Add result of step 7 and step 9.
STEP 11 : Result of steps 1 XOR result of step 9.
STEP 12 : Result of steps 3 XOR result of step 9.
STEP 13 : Result of steps 2 XOR result of step 10.
STEP 14 : Result of steps 4 XOR result of step 10.

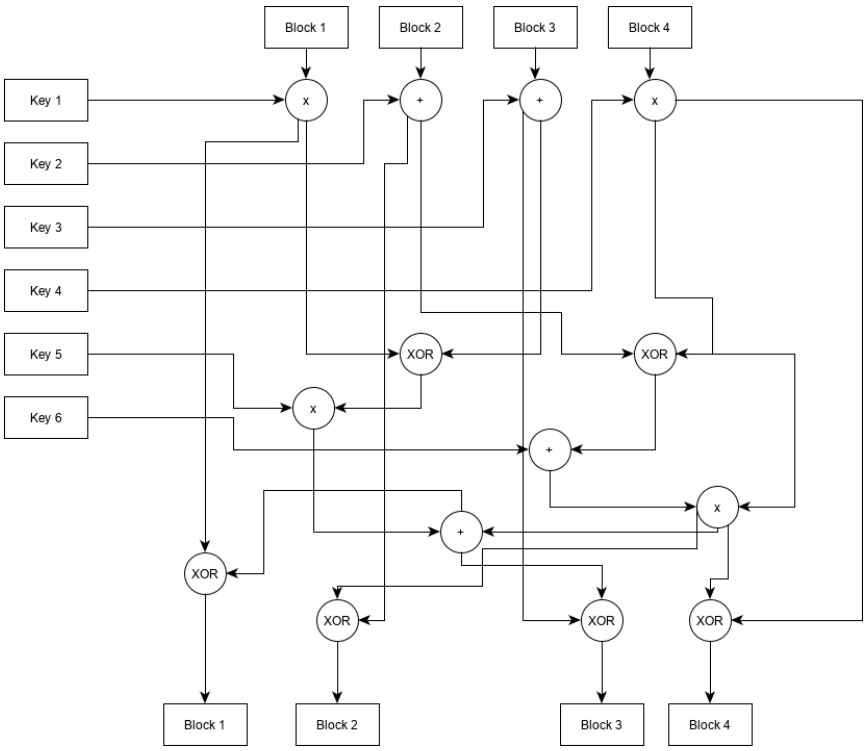
The last round is termed as Output Transformation. It uses 4 16bits sub-keys each. The operations performed in the Output Transformation round are

1. Multiplication Modulo $2^{16}+1$

2. Addition Modulo $2^{16}+1$

Output Transformation Round

STEP 1 : Multiply R1 with Key 49.
STEP 2 : Add R2 and Key 50.
STEP 3 : Add R3 and Key 51.
STEP 4 : Multiply R4 and Key 52.



Serial Execution Approach

In order to execute the code, the input was considered as a text containing **ASCII characters**. All these characters would lie between 1 – 128 only. Thus, all of these characters would take a maximum of **8bits space in the memory**.

Hence, the plain text was appended with extra spaces are added to make it of length which is a **multiple of 8**. Thus, the new text that was formed would occupy a memory size which is a multiple of 64bits. The text was then converted to a binary format and **divided into pieces of 64bits**.

Each piece was taken and divided into blocks of 16bits each and the entire 9 rounds were performed on them. The resulting blocks were concatenated and then appended to an ongoing **cipher text** string. The decryption keys are now generated and the cipher text block is sent for decryption.

Finally, the resulting string from decryption is trimmed and matched with the original plaintext.

Parallel Execution Approach

The coding language used here was Java SE 12. In order to divide the task into various threads, the *ExecutorService* class was called.

In Java, the maximum number of threads that can execute concurrently is the number of processors in the system. Hence, the number of processors was found out by calling *Runtime.getRuntime().availableProcessors()*. IDEA consists of 14 steps that should be carried out sequentially with one result being dependent on the other. Hence, the core processing of the algorithm cannot be altered. In order to parallelize the procedure, the following things were broken down and fed to other threads to execute.

1. Key Generation
2. Input text going through Encryption Process
3. Decryption Key Generation
4. Cipher text going through Decryption Process

An object of the *ExecutorService* class was created, say service, and the maximum number of threads to be created were passed. *ExecutorService* creates a thread pool with the threads in the pool ready to be taken and assigned a task to. Tasks can be submitted or executed by calling the functions, *submit()* and *execute()* respectively. The difference between *submit()* and *execute()* was that *submit()* returns a data type when completed while *execute()* does not.

Sample Usage of the ExecutorService class

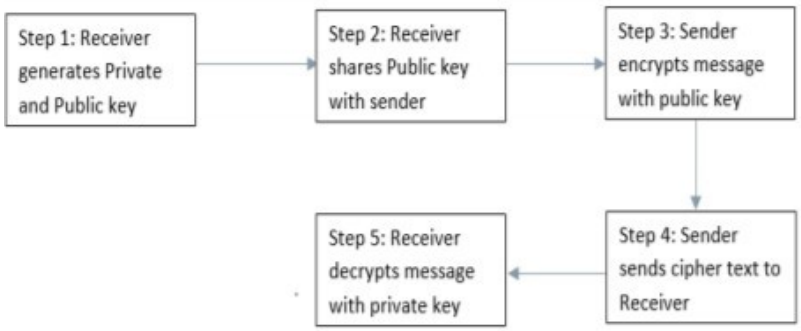
```
class Main {
        public static void main(String[] args) { ExecutorService
                service = new ExecutorService(); List<Future<T>>
                list = new ArrayList<>(); for(int i=0;
                i<numberOfBlocks; i++) {
                        list.add(service.submit(new Task());
                }
        }
}

class Task implements Callable<T>
        { @Override
        public T call() {
                // do something
                return T;
        }
}
```

In the above pseudocode, T can be replaced by any data type. In my implementation of IDEA, I added all the return results called by *service.submit()* to a Map, whose keys are the loop indexes of the iterations where the submit() method is called. Later, all those results are extracted by using the *service.get()* method.

6.2 RSA

PUBLIC KEY BASED CRYPTO ALGORITHMIC APPROACH

Public key based cryptographic algorithms use two kind of keys for encryption and decryption. It is for this reason these algorithms are referred to as Asymmetric Cryptographic Algorithms. For secure data communication between sender and receiver, receiver generates private and public key. Then receiver sends the public key to the sender over a secured medium and requests him to begin the communication. Now, using public key sender encrypts the data and sends the cipher text to receiver. With the help of corresponding private key, receiver decrypts the cipher text and gets the original message. The whole procedure is shown in below figure as a flow diagram.



RSA works on the mathematical concept of factorization of very large number, i.e. finding two large prime numbers whose product is that number. The size of RSA keys is taken large enough to make practically difficult to identify these numbers. This makes factorization to take very long time despite the use of best-known algorithms. The RSA security measure is highly dependent on the size of the key taken which increases the randomness of the determination of factors of the number. It has been assessed that if the key length is of 1024 bits or more, it is nearly impossible to breach the security of RSA encryption even when working with high performance computers. RSA algorithm works in three phases- Key Generation phase, Encryption phase and Decryption phase.

Algorithm

STEP 1 : START

STEP 2 : Choose two large random prime integer numbers x and y (with bit size of 512 or more). STEP 3 : Calculate z as z = x*y refered to as modulus

STEP 4 : Calculate f(n) as f(n)=(x-1) *(y-1)

STEP 5 : Choose an integer w, $1 < w < f(n)$ such that: GCD (w,f(n)) = 1 (where GCD is greatest common denominator)

STEP 6 : Calculate a, $1 < a < f(n)$ such that: $a.w \equiv 1 \pmod{f(n)}$

STEP 7 : STOP

Pseudocode

```
from decimal import Decimal


def gcd(a,b):
    if b==0:
        return a
    else:
        return gcd(b,a%b)
p = int(input('Enter the value of p = '))
q = int(input('Enter the value of q = '))
no = int(input('Enter the value of text = '))
n = p*q
t = (p-1)*(q-1)


for e in range(2,t):
    if gcd(e,t)== 1:
        break
for i in range(1,10):
    x = 1 + i*t
    if x % e == 0:
        d = int(x/e)
        break
ctt = Decimal(0)
ctt =pow(no,e)
ct = ctt % n
dtt = Decimal(0)
dtt = pow(ct,d)
dt = dtt % n
print('n = '+str(n)+' e = '+str(e)+' t = '+str(t)+' d = '+str(d)+' cipher text = '+str(ct)+' decrypted text =
'+str(dt))
```

6.3 Rivest Cipher 5

RC5 is a symmetric key block encryption algorithm designed by Ron Rivest in 1994. It is notable for being simple, fast (on account of using only primitive computer operations like XOR, shift, etc.) and consumes less memory.

Key: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 Plain Text: 00000000 00000000 Cipher Text: EEDBA521 6D8F4B15

RC5 is a block cipher and addresses two-word blocks at a time. Depending on input plain text block size, number of rounds and key size, various instances of RC5 can be defined and each instance is denoted as RC5-w/r/b where w=word size in bits, r=number of rounds and b=key size in bytes.

Algorithm

Step-1: Initialization of constants P and Q.

RC5 makes use of 2 magic constants P and Q whose value is defined by the word size w.

WORD SIZE (BITS)	P (HEXADECIMAL)	Q (HEXADECIMAL)
16	b7e1	9e37
32	b7e15163	9e3779b9
64	b7e151628aed2a6b	9e3779b97f4a7c15

Step-2: Converting secret key K from bytes to words.

Secret key K of size b bytes is used to initialize array L consisting of c words where c = b/u, u = w/8 and w = word size used for that particular instance of RC5. For example, if we choose w=32 bits and Key k is of size 96 bytes then, u=32/8=4, c=b/u=96/4=24.

L is pre initialized to 0 value before adding secret key K to it.

```
for i=b-1 to 0
    L[i/u] = (L[u/i] <<< 8) + K[i]
```

Step-3: Initializing sub-key S.

Sub-key S of size t=2(r+1) is initialized using magic constants P and Q.

```
S[0] = P
for i = 1 to 2(r+1)-1
    S[i] = S[i-1] + Q)
```

Step-4: Sub-key mixing.

The RC5 encryption algorithm uses Sub key S. L is merely, a temporary array formed on the basis of user entered secret key.

Mix in user's secret key with S and L.

```
i = j = 0
A = B = 0
do 3 * max(t, c) times:
    A = S[i] = (S[i] + A + B) <<< 3
    B = L[j] = (L[j] + A + B) <<< (A + B)
    i = (i + 1) % t
    j = (j + 1) % c
```

Step-5: Encryption.

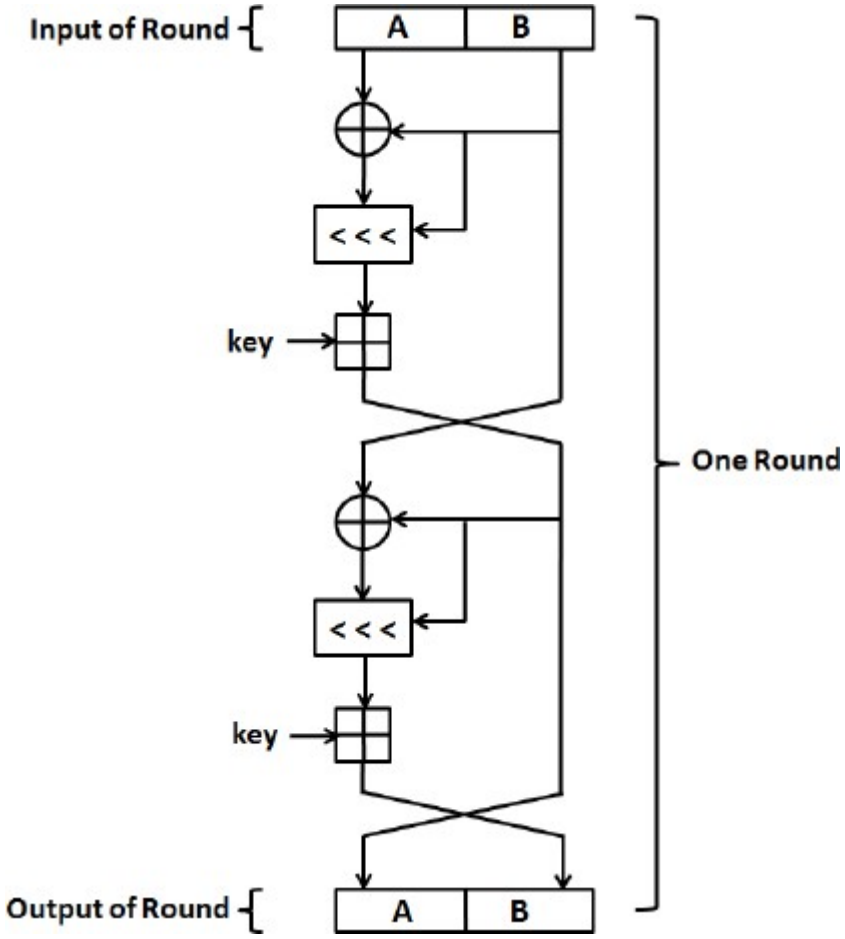
We divide the input plain text block into two registers A and B each of size w bits. After undergoing the encryption process the result of A and B together forms the cipher text block.

```
A = A + S[0]
B = B + S[1]
for i = 1 to r do:
    A = ((A ^ B) <<< B) + S[2 * i]
    B = ((B ^ A) <<< A) + S[2 * i + 1]
return A, B
```

Step-6: Decryption.

```
for i = r down to 1 do:
    B = ((B - S[2 * i + 1]) >>> A) ^ A
    A = ((A - S[2 * i]) >>> B) ^ B
B = B - S[1]
A = A - S[0]
return A, B
```

Block diagram for the one-round in RC5



7. Implementation

7.1. IDEA

4.1.1 Description of Modules/Programs

Class Key is the one storing the key of the encryption\decryption process. The Key object has many functions such as expanding the original key into 52 different sub keys.
Text is the class that stores the instruction. It has functions that divide the text into lengths of 64bits for the algorithm to work on it.
IDEA class performs the main algorithm

4.1.2 Source Code

Serial Approach
Serial_IDEA.java

```
package IDEAAlgorithm.IDEA_Serial;

import IDEAAlgorithm.commons.IDEASequence;
import IDEAAlgorithm.commons.Key;
import IDEAAlgorithm.commons.Utilities;

public class Serial_IDEA {

  private final Utilities utils = new Utilities();
```

```
public static void main(String[] args)
{ Serial_IDEA object = new Serial_IDEA();
String text = "Hello World"; // sample text

 System.out.println("Starting Encryption Process with Plain Text = " + text);

 /*
  * Array containing the cipher text and time taken
  * [0] => cipher text
  * [1] => Time taken
  */
 String[] returnedCipher =
 object.doIDEAEncryption(text); String
 cipher = returnedCipher[0];
 long encrypt = Long.parseLong(returnedCipher[1]);
 System.out.printf("Cipher text : %s\n Time taken to encrypt %d\n", cipher, encrypt);

 /*
  * Array containing the plain text and time taken
  * [0] => plain text
  * [1] => Time taken
  */
 String[] returnedPlain =
 object.doIDEADecryption(cipher); String
 pt = returnedPlain[0];
 long decrypt = Long.parseLong(returnedPlain[1]);
 System.out.printf("Decrypted text : %s\n Time taken to decrypt %d\n", pt, decrypt);
}

public String[]
 doIDEAEncryption(String PlainText)
 { KeyGenerator generator = new
 KeyGenerator(); long start =
 System.currentTimeMillis();
Key key =
 generator.generateKeySet(new
 Key().getOriginalKeyString(), true);
 Text text = new Text(PlainText,
 true);
 return new String[]{
    new IDEASequence(key,
    text).performIDEASequence().trim(),
    Long.toString(System.currentTimeMillis() -
    start)
 };
}

public String[] doIDEADecryption(String CipherText) {
 long start =
 System.currentTimeMillis();
 Text text = new
 Text(CipherText, false); Key
 key = new Key();
 KeyGenerator generator = new KeyGenerator();
```

```
    key=generator.generateKeySet(key.getOriginalKe
    yString(), false); return new String[] {
        utils.getDecryptedString(new                IDEASequence(key,
        text).performIDEASequence()),
        Long.toString(System.currentTimeMillis() - start)
    };
  }
}
```

Key Generator.java

```
package IDEAAlgorithm.IDEA_Serial;

import IDEAAlgorithm.commons.Key;
import IDEAAlgorithm.commons.Utilities;

import java.util.ArrayList;

public class KeyGenerator {

  public Key generateKeySet(String originalKey,
    boolean isEncryption) { originalKey =
    rounder(originalKey);
    String eightRounds =
    originalKey.substring(0, 768); String
    halfRound =
    originalKey.substring(768);

    ArrayList<String[]> myList = new
    ArrayList<>(); String[] temp;
    int v = 0;
    for (int i = 0; i < 8;
      i++) { temp =
      new String[6]; for
      (int j = 0; j < 6; j+
      +) {
        temp[j] =
        eightRounds.substring(v, v +
        16); v += 16;
      }
      myList.add(temp);
    }

    temp = new
    String[4]; v = 0;
    for (int i = 0; i < 4; i++) {
      temp[i] =
      halfRound.substring(v, v +
      16); v += 16;
    }
    myList.add(temp);

    if (!isEncryption) {
```

```
    myList = performKeyReversal(myList);
   }

   return new Key(myList);
 }
 private String rounder(String originalKey) { String tempString = originalKey;
   for (int i = 1; i <= 6; i++) {
     originalKey = circularLeftShiftBy25(originalKey);
     if (i != 6) {
       tempString = tempString.concat(originalKey);
     } else {
       tempString = tempString.concat(originalKey.substring(0, 64));
     }
   }
   return tempString;
 }

 private String
   circularLeftShiftBy25(String number)
   { String finalNumber = "";
   for (int i = 0; i < 25; i++) {
     finalNumber = finalNumber.concat(Character.toString(number.charAt(i)));
   }
   finalNumber = number.substring(25).concat(finalNumber);
   return finalNumber;
 }

 private final Utilities utilities = new Utilities();

 private ArrayList<String[]>
   performKeyReversal(ArrayList<String[]> myList)
   { ArrayList<String[]> newList = new ArrayList<>();
   String[][] store = new String[9][6];
   for (int i = 0; i < 9; i++) {
     System.arraycopy(myList.get(i), 0, store[i], 0, myList.get(i).length);
   }

   for (int i = 0; i < 9; i++) {
     if (i != 8) {
       newList.a
       dd(
           new String[]
               { utilities.findInverse(store
               [8 - i][0]),
               utilities.findNegative(store[
               8 - i][1]),
               utilities.findNegative(store[
               8 - i][2]),
               utilities.findInverse(store[8
               - i][3]), store[8 - i - 1][4],
               store[8 - i - 1][5]
           }
       );
     } else
       { newList
```

```
      .add(
          new String[]
              { utilities.findInverse(store
              [8 - i][0]),
              utilities.findNegative(store[
              8 - i][1]),
              utilities.findNegative(store[
              8 - i][2]),
              utilities.findInverse(store[8
              - i][3]),
          }
        );
      }
    }

    return newList;
  }
}
```

Text,java

```
package IDEAAlgorithm.IDEA_Serial; import IDEAAlgorithm.commons.Utilities; import
java.util.ArrayList;
public class Text {
  private String text;
  private ArrayList<String[]> textBlocks;

  public Text(String text, boolean isPlainText) {
    this.text = text;
    textBlocks = new ArrayList<>();
    if (isPlainText)
      { setText();
      formPlainTextBl
      ocks();
    } else
      { formCipherText
      Blocks();
    }
  }

  private void setText() {
    int length = text.length() % 8;
    for (int i = length; i < 8; i++) {
      text = text.concat(" ");
    }
  }

  private void
    formPlainTextBlocks()
    { Utilities utils = new
    Utilities();
    int i = 0;
    String[]
    temp;
    // LOGIC: the text blocks have to be divided into 2 each
```

```
    // LOGIC: hence, if the length is not divisible by 2, add an extra space to make it even
    while (i <
      text.length())
      { temp = new
      String[4];
      temp[0] =
      utils.decimalToBinary8(text.charAt(i));
      temp[1] =
      utils.decimalToBinary8(text.charAt(i +
      1)); temp[2] =
      utils.decimalToBinary8(text.charAt(i +
      2)); temp[3] =
      utils.decimalToBinary8(text.charAt(i +
      3));
//     System.out.println(temp.length());
      textBlocks.add(t
      emp); i += 4;
    }
  }

  private void
    formCipherTextBlocks()
    { String str;
    int i = 0;
    while (i < text.length()) {
      str = text.substring(i, i + 32);
      textBlocks.a
          dd( new
          String[]{
              str.substring(0, 8),
              str.substring(8, 16),
              str.substring(16, 24),
              str.substring(24, 32)
          }
      );
      i += 32;
    }
  }
  public ArrayList<String[]> getTextBlocks() {
    return textBlocks;
  }
}
```

Parallel Approach

Parallel_IDEA.java

```
package IDEAAlgorithm.IDEA_Parallel;

import IDEAAlgorithm.commons.Key;
import IDEAAlgorithm.commons.Utilities;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.concurrent.Callable;
```

```
importjava.util.concurrent.ExecutinException;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.concurrent.Future;

@SuppressWarnings("Duplicates")
public class Parallel_IDEA {

  private final static Utilities utils = new Utilities();
  private final int numberOfCores = Runtime.getRuntime().availableProcessors();
  private Key key = new Key();

  public static void main(String[]
    args) { Parallel_IDEA object =
    new Parallel_IDEA(); String
    plainText = "Hello World";

    /*
     * Array containing the cipher text and time taken
     * [0] => cipher text
     * [1] => Time taken
     */
    String[] returnedCipher =
    object.doEncryptionParallel(plainText); String
    cipher = returnedCipher[0];
    long encrypt = Long.parseLong(returnedCipher[1]);
    System.out.printf("Cipher text : %s\n Time taken to encrypt %d\n", cipher, encrypt);

    /*
     * Array containing the plain text and time taken
     * [0] => plain text
     * [1] => Time taken
     */
    String[] returnedPlain =
    object.doDecryptionParallel(cipher); String pt
    = returnedPlain[0];
    long decrypt = Long.parseLong(returnedPlain[1]);
    System.out.printf("Decrypted text : %s\n Time taken to decrypt %d\n", pt, decrypt);

    // checking for equality of original text and decrypted text
    if (pt.equals(plainText)) {
      System.out.println("Successful Encryption and Decryption");
    }
  }

  public String[] doEncryptionParallel(String plainText) {
    try {
    // form key
      long init = System.currentTimeMillis();
      ExecutorService service = Executors.newFixedThreadPool(numberOfCores);
      key = new KeyMaker().makeKey(key);
      int length = plainText.length() % 8;
      if (length != 0) {
        for (int i = length; i < 8;
          i++) { plainText =
          plainText.concat(" ");
```

```
  }
}
HashMap<Integer, Future<String>> PlainTextMap = new HashMap<>();
int i;
for (i = 0; i < plainText.length(); i += 8) {
  for (int j = 0; j < 8; j++) {
    PlainTextMap.put((j + i), service.submit(new TextParallel(plainText.charAt(i + j), utils)));
  }
}
long time =
System.currentTimeMillis() - init;
ArrayList<String[]> textBlocks =
new ArrayList<>(); for (i = 0; i <
plainText.length(); i += 4) {
  textBlocks.add(
      new String[]
          { PlainTextMap.get(i)
          .get(),
          PlainTextMap.get(i +
          1).get(),
          PlainTextMap.get(i +
          2).get(),
          PlainTextMap.get(i +
          3).get()
      }
  );

}
service.shutdown();
PlainTextMap.clear(); // for garbage collection
System.gc();
long checkPoint1 = System.currentTimeMillis();
service =
Executors.newFixedThreadPool(numberOfCores);
ArrayList<String[]> keySet = key.getKeyList(); //
getting keys HashMap<Integer, Future<String>>
CipherTextMap = new HashMap<>(); for (i = 0; i <
textBlocks.size(); i += 2) {
  CipherTextMap.put(i, service.submit(new
      Parallel_IDEA_Task( textBlocks.get(i),
      textBlocks.get(
      i + 1), keySet,
      utils
  )));
}
String cipher = "";
for (i = 0; i < textBlocks.size(); i += 2) {
  cipher = cipher.concat(CipherTextMap.get(i).get());
}
time += System.currentTimeMillis() - checkPoint1;
return new
    String[]
    { cipher,
    Long.toString(
    time)
};
```

```
  }
  catch (InterruptedException |
    ExecutionException e)
    { e.printStackTrace();
  }
  return new String[]{};
}

public String[] doDecryptionParallel(String cipherText) {
  try {
    ExecutorService service = Executors.newFixedThreadPool(numberOfCores);
    String str;
    String[] topArray, bottomArray;
    HashMap<Integer, Future<String>> TextMap = new HashMap<>();
    long init = System.currentTimeMillis();
    key = new KeyMaker().invertKey(key.getKeyList());
    for (int i = 0; i <
      cipherText.length(); i += 64)
      { str = cipherText.substring(i, i
      + 64); topArray = new String[]{
          str.substring(0, 8),
          str.substring(8, 16),
          str.substring(16, 24),
          str.substring(24, 32)
      };
      bottomArray = new
          String[]
          { str.substring(32,
          40),
          str.substring(40, 48),
          str.substring(48, 56),
          str.substring(56, 64)
      };
      TextMap.put(i, service.submit(new
          Parallel_IDEA_Task( topArray,
          bottomArra
          y,
          key.getKey
          List(), utils
      )));
    }
    str = "";
    for (int i = 0; i <
      cipherText.length(); i += 64)
      { str =
      str.concat(TextMap.get(i).get());
    }
    long time =
    System.currentTimeMillis() - init;
    service.shutdown();
    return new String[]{
        utils.getDecryptedStri
        ng(str),
        Long.toString(time)
    };
  } catch (InterruptedException |
```

```
    ExecutionException e)
      { e.printStackTrace();
    }
    return new String[]{};
  }
}

class Parallel_IDEA_Task implements Callable<String> {

  private String[]
  topArray; private
  String[] bottomArray;
  private static ArrayList<String[]> keySet;
  private Utilities utilities;

  Parallel_IDEA_Task(String[] topArray, String[] bottomArray, ArrayList<String[]> keySet, Utilities
  utilities) {
    this.topArray = topArray;
    this.bottomArray =
    bottomArray;
    Parallel_IDEA_Task.keyS
    et = keySet; this.utilities =
    utilities;
  }

  @Override
  public String
    call() { String
    p1, p2, p3, p4;
    p1 = topArray[0] +
    topArray[1]; p2 =
    topArray[2] +
    topArray[3];
    p3 = bottomArray[0] + bottomArray[1];
    p4 = bottomArray[2] + bottomArray[3];
    return utilities.doIDEASequenceLoop(p1, p2, p3, p4, utilities, Parallel_IDEA_Task.keySet);
  }
}

class TextParallel implements Callable<String> {

  private char text;
  private Utilities
  utilities;

  TextParallel(char text, Utilities utilities) {
    this.text = text;
    this.utilities = utilities;
  }

  @Override
  public String call() {
    return utilities.decimalToBinary8(text);
  }
}
```

KeyMaker.java

```
package IDEAAlgorithm.IDEA_Parallel;

import IDEAAlgorithm.commons.Key;
import IDEAAlgorithm.commons.Utilities;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.concurrent.Callable;
import
java.util.concurrent.ExecutionExcep
tion; import
java.util.concurrent.ExecutorService
; import
java.util.concurrent.Executors;
import java.util.concurrent.Future;

class KeyMaker {
  Key makeKey(Key key) {
    try {
      int numberOfCores = 8;
      ExecutorService service =
      Executors.newFixedThreadPool(numberOfCores);
      String keyString = key.getOriginalKeyString();
      HashMap<Integer, Future<String>> taskMap = new HashMap<>();
      for (int i = 0; i < 7; i++) {
        taskMap.put(i, service.submit(new KeyMakerTask(keyString, 25 * i)));
      }
      String megaKey
      = ""; for (int i =
      0; i < 7; i++) {
        megaKey = megaKey.concat(taskMap.get(i).get());
      }

      ArrayList<String[]> keyList = new
      ArrayList<>(); String[] temp;
      int index = 0, limit;
      for (int i = 0; i < 9;
        i++) { limit = i ==
        8 ? 4 : 6;  temp =
        new String[limit];
        for (int j = 0; j < limit; j++) {
          temp[j] = megaKey.substring(index,
          index + 16); index += 16;
        }
        keyList.add(temp);}
      service.shutdown
      (); key = new
      Key(keyList);
    } catch (InterruptedException |
      ExecutionException e)
      { e.printStackTrace();
    }
    return key;
  }
```

```
  Key invertKey(ArrayList<String[]> myList) {
    try {
      for (int i = 0; i < 9; i++) {
        System.arraycopy(myList.get(i), 0, KeyInverterTask.store[i], 0, myList.get(i).length);
      }

      ExecutorService service =

      Executors.newFixedThreadPool(Runtime.getRuntime().availableProcessors())

      ; HashMap<Integer, Future<String[]>> taskMap = new HashMap<>();
      for (int i = 0; i < 9; i++) {
        taskMap.put(i, service.submit(new KeyInverterTask(i)));
      }

      ArrayList<String[]> newList = new ArrayList<>();
      for (int i = 0; i < 9; i++)
        { newList.add(taskMap.g
        et(i).get());
      }
      service.shutdown();
      return new Key(newList);
    } catch (InterruptedException |
      ExecutionException e)
      { e.printStackTrace();
    }
    return null;
  }
}

class KeyMakerTask implements Callable<String> {

  private String
  data; private
  int times;

  KeyMakerTask(String data, int times) {
    this.data = data;
    this.times = times;
  }

  @Overr
     ide
   public String
     call() { if
    (times == 0)
    { return data;
    }
    if (times > 128) {
      times = times % 128;
    }
    String
    finalNumber = "";
    int i;
    for (i = 0; i < times; i++) {
      finalNumber = finalNumber.concat(Character.toString(data.charAt(i)));
```

```
    }
    finalNumber =
    data.substring(times).concat(finalNumber);
    return finalNumber;
  }
}
class KeyInverterTask implements Callable<String[]> {
  private int i;
  private final Utilities utilities = new Utilities();
  static String[][] store = new String[9][6];

  KeyInverterTask(int i) {
    this.i = i;
  }

  @Override
  public String[] call() {
    if (i != 8) {
      return new String[]
          { utilities.findInverse(KeyInverterTask.
          store[8 - i][0]),
          utilities.findNegative(KeyInverterTask.
          store[8 - i][1]),
          utilities.findNegative(KeyInverterTask.
          store[8 - i][2]),
          utilities.findInverse(KeyInverterTask.st
          ore[8 - i][3]), KeyInverterTask.store[8 -
          i - 1][4], KeyInverterTask.store[8 - i -
          1][5]
      };
    } else {
      return new String[]
          { utilities.findInverse(KeyInverterTask.
          store[8 - i][0]),
          utilities.findNegative(KeyInverterTask.
          store[8 - i][1]),
          utilities.findNegative(KeyInverterTask.
          store[8 - i][2]),
          utilities.findInverse(KeyInverterTask.st
          ore[8 - i][3]),
      };
    }
  }
}
```

Common Classes

Utilitites.java

```
package IDEAAlgorithm.commons;

import java.util.ArrayList;

public class Utilities {
  public String decimalToBinary8(int
    decimalNumber) { String decimal =
    Integer.toBinaryString(decimalNumber);
```

```
int l = decimal.length();
  if (l < 8) {
    for (int i = 0; i <= 7 - l; i++) {
                                    //

  System.out.println("adding
                                    0");
                          decimal =
          "0".concat(decimal);
    }
  }
  return decimal;
}

private String decimalToBinary16(long
  decimalNumber) { String decimal =
  Long.toBinaryString(decimalNumber);
  int l = decimal.length();
  if (l < 16) {
    for (int i = 0; i <= 15 - l;
      i++) { decimal =
      "0".concat(decimal);
    }
  }
  return decimal;
}

private int binaryToDecimal16(String binaryString) {
  int decimal = 0;
  for (int i = 15; i >= 0; i--) {
    decimal += ((int) (binaryString.charAt(15 - i)) - 48) * (int) Math.pow(2, i);
  }
  return decimal;
}

private int binaryToDecimal8(String binaryString) {
  int decimal = 0;
  for (int i = 7; i >= 0; i--) {
    decimal += ((int) (binaryString.charAt(7 - i)) - 48) * (int) Math.pow(2, i);
  }
  return decimal;
}

private String MultiplicationModulo(String x, String y) {
  long n1 = binaryToDecimal16(x);
  long n2 = binaryToDecimal16(y);
  return decimalToBinary16(((n1 * n2) % 65537));

}

private String AdditionModulo(String x, String y)  {
  int n1 = binaryToDecimal16(x);
  int n2 = binaryToDecimal16(y);
  return decimalToBinary16(((n1 + n2) % 65536));
}
```

```
private String XOR(String
  x, String y) { String xor =
  "";
for (int i = 0; i < 16; i++) {
    if (x.charAt(i) !=
      y.charAt(i)) { xor =
      xor.concat("1");
    } else {
      xor = xor.concat("0");
    }
  }
  return xor;
}


public String
  findInverse(String x)  { long
  decimal =
  binaryToDecimal16(x);
  decimal = decimal %
  65537;
  for (int i = 1; i < 65537; i++) {
if ((decimal * i) % 65537 ==
                1) {
return decimalToBinary16(i);
    }
  }
  return decimalToBinary16(1L);
}

public String
  findNegative(String str1)
  { long x =
  binaryToDecimal16(str1);
  String temp =
  Long.toBinaryString(-x);
  return temp.substring(temp.length() - 16);
}

public String
  getDecryptedString(String cipher)
  { String x, temp = "";
  for (int i = 0; i <
    cipher.length(); i += 8) { x =
    cipher.substring(i, i + 8);
    temp = temp.concat(Character.toString((char)binaryToDecimal8(x)));
  }
  return temp;
}

public String doIDEASequenceLoop(String p1, String p2, String p3, String p4, Utilities utils,
ArrayList<String[]> keySet) {

  String result1, result2, result3, result4, result5, result6, result7,
      result8, result9, result10, result11, result12, result13, result14;
```

```
String[] currentKeySet;
for (int j=1; j<=8; j++)
  { currentKeySet =
  keySet.get(j - 1);

  // step 1
  result1 = utils.MultiplicationModulo(p1, currentKeySet[0]);

  // step 2
  result2 = utils.AdditionModulo(p2, currentKeySet[1]);

  // step 3
  result3 = utils.AdditionModulo(p3, currentKeySet[2]);

  // step 4
  result4 = utils.MultiplicationModulo(p4, currentKeySet[3]);

  // step 5
  result5 = utils.XOR(result1, result3);

  // step 6
  result6 = utils.XOR(result2, result4);
  // step 7
  result7 = utils.MultiplicationModulo(result5, currentKeySet[4]);

  // step 8
  result8 = utils.AdditionModulo(result6, result7);

  // step 9
  result9 = utils.MultiplicationModulo(result8, currentKeySet[5]);

  // step 10
  result10 = utils.AdditionModulo(result7, result9);

  // step 11
  result11 = utils.XOR(result1, result9);

  // step 12
  result12 = utils.XOR(result3, result9);

  // step 13
  result13 = utils.XOR(result2, result10);

  // step 14
  result14 = utils.XOR(result4, result10);

currentKeySet = keySet.get(8);
p1 = utils.MultiplicationModulo(p1,
currentKeySet[0]); p2 =
utils.AdditionModulo(p2,
currentKeySet[1]);
p3 = utils.AdditionModulo(p3, currentKeySet[2]);
p4 = utils.MultiplicationModulo(p4, currentKeySet[3]);
// System.out.println(p1 + " - " + p2 + " - " + p3 + " - " + p4);
// converting back to decimal and adding the corresponding ASCII to cipher text
// keep the text as binary only
```

```
    return p1 + p2 + p3 + p4;
  }
}
```

Key.java

```
package IDEAAlgorithm.commons;

import java.util.ArrayList;

public class Key {
  private ArrayList<String[]> keyList;

  public Key() {
    keyList = new ArrayList<>();
  }

  public Key(ArrayList<String[]> keyList) {
    this.keyList = keyList;
  }

  public ArrayList<String[]> getKeyList() {
    return keyList;
  public void setKeyList(ArrayList<String[]> keyList) {
    this.keyList = keyList;
  }

  public String getOriginalKeyString() {
    return
"10010111011111000000111001100010101111100000111100111000100110101110001110111001
1101010000001000010101000010100100001101111110101";
  }
}
```

IDEASequence.java

```
package

IDEAAlgorithm.commons;

import

IDEAAlgorithm.IDEA_Serial

.Text; import

java.util.ArrayList;

public class IDEASequence {
  private final Utilities utils = new Utilities();
  private Key
  key; private
  Text text;
```

```
  public IDEASequence(Key key, Text text) {
    this.key = key;
    this.text = text;
  }

  public String
    performIDEASequence()
    { String cipherText = "";
    ArrayList<String[]> keySet =
    key.getKeyList();
    ArrayList<String[]> plain =
    text.getTextBlocks(); String p1, p2,
    p3, p4;
    String resultBack;
    for (int i = 0; i <
      plain.size(); i += 2) { p1 =
      plain.get(i)[0] +
      plain.get(i)[1];
      p2 = plain.get(i)[2] + plain.get(i)[3];
      p3 = plain.get(i + 1)[0] +
      plain.get(i + 1)[1]; p4 =
      plain.get(i + 1)[2] + plain.get(i
      + 1)[3];
//     System.out.println(p1 + " - " + p2 + " - " + p3 + " - " + p4);
      resultBack = utils.doIDEASequenceLoop(p1, p2, p3, p4, utils, keySet);
//     System.out.println("Before half round : ");
//     System.out.println(p1 + " - " + p2 + " - " + p3 + " - " + p4);
      // half round
      cipherText = cipherText.concat(resultBack);
    }
    return cipherText;
  }
}
```

7.2 RSA

1. prime_num_cal(n): function to calculate large prime numbers
2. mod_calculator(num,mod): calculating the modulus for large numbers
3. gcd(a,b) : calculating gcd using Euclidian method
4. modInverse(a, m) : calculation of modulo inverse using extended Euclidian method

1) RSA SERIAL CODE IN PYTHON:

```
#Implementation of RSA algorithm to generate large prime numbers

import time

# function to calculate large prime numbers using Seive of eratosthenes algorithm def
prime_num_cal(n):

    # creating a list of numbers substituted with "True" in place
    range_of_n = [True for x in range(n+1)]

    #Smallest prime number
    p=2
    while(p*p <= n):

        # If not 'false' then number is not prime if
        (range_of_n[p] == True):

            for i in range(p*p , n+1 , p):

                #Marking 'True' for the multiple of 'P'
                range_of_n[i]=False
        p=p+1

    prime_numbers=[]

    for i in range(int(n*0.5),n+1):

        if(range_of_n[i]==True):

            prime_numbers.append(i)

    return prime_numbers

# calculating the modulus for large numbers def

mod_calculator(num,mod):

    residual = 0
```

```
temp = str(num)
for i in range(0,len(temp)):
      residual = (residual*10 + int(temp[i]))%mod

return residual
```

```
# calculating gcd using euclidian method

  def gcd(a,b):
      if(a==0):
          return b return

      gcd(b%a,a)

# calculation of modulo inverse using extended euclidian method

def modInverse(a, m) :
      m0 = m
      y = 0
      x = 1

      if (m == 1) :
          return 0

      while (a > 1) :

          q = a // m t

          = m

          m = a %
          m a = t
          t = y

          y = x - q * y x
          = t

      if (x < 0) :
          x = x + m0

      return x

# generating prime numbers

prime_num1 = prime_num_cal(200)[-
1] prime_num2 = prime_num_cal(300)
[-1]

# calculating 'n' and euler's totient function n=

prime_num1 * prime_num2
eulers_totient = (prime_num1-1) * (prime_num2-1)

# calculating public key for encryption
```

```
for i in range(eulers_totient,int(eulers_totient*0.5),-1): if (i
      %2==1):
            if(gcd(eulers_totient,i)==1):
                  public_key=i
                  break

# calculating private key for decryption

private_key = (modInverse(public_key,eulers_totient))
```

```
# creating dictionary of alphabets and there resp positions

alpha = [chr(x) for x in range(ord('a'), ord('z') + 1)] pos = [x
for x in range(1,27)]
enc_dict = {k:v for (k,v) in zip(alpha,pos)}
dec_dict = {k:v for (k,v) in zip(pos,alpha)}
enc_dict[' ']= 0
dec_dict[0]=' ' #print(enc_dict,"\
n",dec_dict)

# user section initial =
time.time()

inp = "trump killed kennedy and usa dropped the nuclear bomb"
position,enc_text,dec_text = [],[],[]

for i in string:
    position.append(enc_dict[i])

for i in position:
    enc_text.append(str(mod_calculator(pow(i,public_key),n)))

print("\nencrypted text : "+".join(enc_text)+"\n") for i in

enc_text:
    dec_text.append(dec_dict[mod_calculator(pow(int(i),private_key),n)])

print("decrypted text : "+".join(dec_text))

end = time.time()
print("\ntime elapsed ",end-initial)
```

2) PARALLEL RSA IMPLEMENTATION IN PYTHON USING MULTIPROCESSING:

```
 #Implementation of RSA algorithm to generate large prime numbers
import multiprocessing
import time

# function to calculate large prime numbers using Seive of eratosthenes algorithm def
prime_num_cal(n):

    # creating a list of numbers substituted with "True" in place
    range_of_n = [True for x in range(n+1)]

    #Smallest prime number
    p=2
    while(p*p <= n):

        # If not 'false' then number is not prime if
        (range_of_n[p] == True):
```

```
        for i in range(p*p , n+1 , p):

            #Marking 'True' for the multiple of 'P'
            range_of_n[i]=False
    p=p+1

prime_numbers=[]
```

```
    for i in range(int(n*0.5),n+1):

        if(range_of_n[i]==True):

            prime_numbers.append(i)

    return prime_numbers

# calculating the modulus for large numbers def

mod_calculator(num,mod):

    residual = 0
    temp = str(num)
    for i in range(0,len(temp)):
        residual = (residual*10 + int(temp[i]))%mod

    return residual

# calculating gcd using euclidian method

  def gcd(a,b):
     if(a==0):
        return b return

    gcd(b%a,a)

# calculation of modulo inverse using extended euclidian method

def modInverse(a, m) :
    m0 = m
    y = 0
    x = 1

    if (m == 1) :
        return 0

    while (a > 1) :

        q = a // m t

        = m

        m = a %
        m a = t
        t = y
```

```
            y = x - q * y x
            = t

        if (x < 0) :
            x = x + m0

        return x

# generating prime numbers

prime_num1 = prime_num_cal(200)[-
1] prime_num2 = prime_num_cal(300)
[-1]
```

```
# calculating 'n' and euler's totient function

n= prime_num1 * prime_num2
eulers_totient = (prime_num1-1) * (prime_num2-1) #

calculating public key for encryption

for i in range(eulers_totient,int(eulers_totient*0.5),-1): if (i
    %2==1):
        if(gcd(eulers_totient,i)==1):
            public_key=i
            break

# calculating private key for decryption

private_key = (modInverse(public_key,eulers_totient))

# creating dictionary of alphabets and there resp positions

alpha = [chr(x) for x in range(ord('a'), ord('z') + 1)] pos = [x
for x in range(1,27)]
enc_dict = {k:v for (k,v) in zip(alpha,pos)}
dec_dict = {k:v for (k,v) in zip(pos,alpha)}
enc_dict[' ']= 0
dec_dict[0]=' ' #print(enc_dict,"\
n",dec_dict) def
mod_calculator_par(num):

    residual = 0
    temp = str(num)
    for i in range(0,len(temp)):
        residual = (residual*10 + int(temp[i]))%n

    return residual

inp = "trump killed kennedy and usa dropped the nuclear bomb" if_
name_=='_main_':
    # user section initial =
    time.time()


    string = list(inp) position_dec,position_enc,enc_text,dec_text =
    [],[],[],[]

    for i in string:
        position_dec.append(enc_dict[i])

    position_dec = [pow(x,public_key) for x in position_dec]
```

```
p_obj = multiprocessing.Pool()

enc_text = p_obj.map(mod_calculator_par,position_dec)
p_obj.close()
p_obj.join()

dup_enc_text = []
dup_enc_text = [str(x) for x in enc_text]
print("encrypted text : "+".join(dup_enc_text))
```

```
position_enc = [pow(x,private_key) for x in enc_text]

p_obj2 = multiprocessing.Pool()

dec_text = p_obj2.map(mod_calculator_par,position_enc)
p_obj2.close()
p_obj2.join()
dec_text1=[]

for text in dec_text:
      dec_text1.append(dec_dict[text])
print("decrypted text : "+ ".join(dec_text1))

end = time.time()
print("\ntime elapsed ",end-initial)
```

7.3 Rivest Cipher 5

The RC5_SETUP() function mixes the private key and the public key.
RC5_ENCRYPT() function encrypts the plaintext and ciphertext is formed.
RC5_DECRYPT() function decrypts the ciphertext and the output is plaintext.
The main() function stores the input plain text and calls the above 3 functions.

Serial Approach

```
#include   <stdio.h>
#include<string.h>
#include<stdlib.h>
#include<omp.h>
typedef unsigned int WORD;
#define w       32                  /* word size in bits                    */
#define r       12                  /* number of rounds                     */
#define b       16                  /* number of bytes in key               */
#define c       4                   /* number words in key = ceil(8*b/w)*/
#define t       26                  /* size of table t = 2*(r+1) words */ WORD
S[t];
WORD P = 0xb7e15163, Q = 0x9e3779b9;//p and q constants are initialised
#define ROTL(x,y) (((x)<<(y&(w-1))) | ((x)>>(w-(y&(w-1)))))
#define ROTR(x,y) (((x)>>(y&(w-1))) | ((x)<<(w-(y&(w-1)))))

void RC5_ENCRYPT(WORD *pt, WORD *ct) /* 2 WORD input pt/output ct   */
{ WORD i, A=pt[0]+S[0], B=pt[1]+S[1];//The plain text blocks A and b by adding S[0] and S[1] for
   (i=1; i<=r; i++)//entire procedure is repeated r times.
      { A = ROTL(A^B,B)+S[2*i];
        B = ROTL(B^A,A)
        +S[2*i+1];
      }
   ct[0] = A; ct[1] = B;
}

void RC5_DECRYPT(WORD *ct, WORD *pt) /* 2 WORD input ct/output pt     */
{ WORD i, B=ct[1], A=ct[0];
   for (i=r; i>0; i--)
     { B = ROTR(B-S[2*i+1],A)^A;
       A = ROTR(A-S[2*i],B)^B;
      }
   pt[1] = B-S[1]; pt[0] = A-S[0];
}

void RC5_SETUP(unsigned char *K) /* secret input key K[0...b-1]              */
{   WORD i, j, k, u=w/8, A, B, L[c];
```

```
    /* Initialize L, then S, then mix key into S */
    for (i=b-1,L[c-1]=0; i!=-1; i--) L[i/u] = (L[i/u]<<8)+K[i];
                        for (S[0]=P,i=1; i<t; i++) S[i] = S[i-
1]+Q;//Sub key is initialised using magic constants p and q
    for (A=B=i=j=k=0; k<3*t; k++,i=(i+1)%t,j=(j+1)%c)       /*L is an array formed on the basis of
 user entered secret key.S and L are mixed with user secret key. 3*t > 3*c */
       { A = S[i] = ROTL(S[i]+(A+B),3);
         B = L[j] = ROTL(L[j]+(A+B),(A+B));
void main()
{
     WORD i,j,pt1[2],pt2[2], ct[2]={0,0};
     unsigned char key[b];
     if (sizeof(WORD)!=4)
          printf("RC5 error: WORD has %d bytes.\n",sizeof(WORD));
     printf("RC5-32/12/16 encryption and decryption:\n");
     for(i=0;i<3;i++)
     {
                    if(i==0)
                    { pt1[0]="Hello";pt1[1]="Parag";}
                    if(i==1)
                    { pt1[0]="Hello";pt1[1]="Manis";
                    } if(i==2)
                    { pt1[0]="Hello";pt1[1]="Nikil";}
                    for(j=0;j<b;j++)
                         key[j]=ct[0]%(255-j);//User secret key is zero.
                    RC5_SETUP(key);
                    RC5_ENCRYPT(pt1,
                    ct);
                    RC5_DECRYPT(ct,pt
                    2);
                    printf("\n key = ");
                    for(j=0;j<b;j++)
                         printf("%.2X ",key[j]);
                                        printf("\nEncryption\n      plaintext     %s   %s   ---
> ciphertext %.8X %.8X \n",pt1[0],pt1[1],ct[0],ct[1]);
                                    printf("\nDecryption\n      ciphertext     %.8X  %.8X  ---
> plaintext %s %s \n",ct[0],ct[1],pt2[0],pt2[1]);
                    if(pt1[0]!=pt2[0] || pt1[1]!=pt2[1])
                         printf("Decryption Error!");
     }
}
```

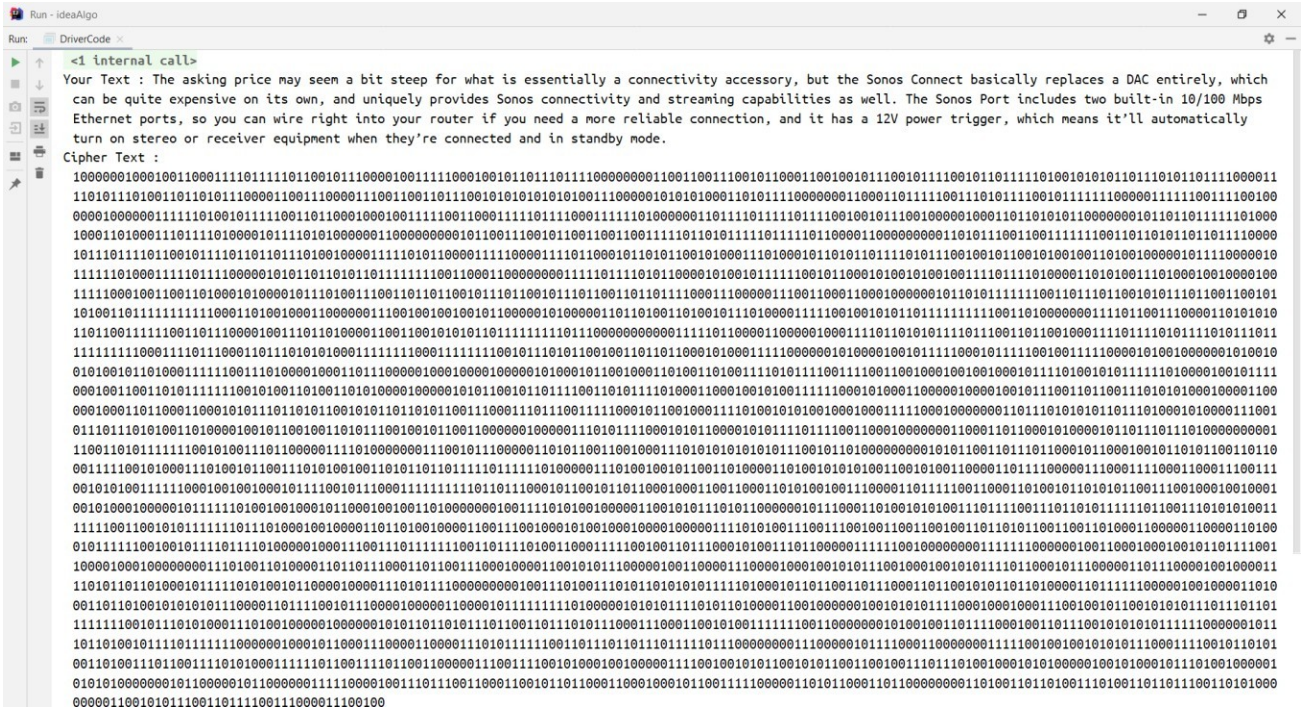
Parallel Approach

```
#include   <stdio.h>
#include<string.h>
#include<stdlib.h>
#include<omp.h>
typedef unsigned int WORD;
```

```
#define w        32                    /* word size in bits                    */
#define r        12                    /* number of rounds                     */
#define b        16                    /* number of bytes in key               */
#define c        4                     /* number words in key = ceil(8*b/w)*/
#define t        26                    /* size of table S = 2*(r+1) words */
WORD S[t];
WORD P = 0xb7e15163, Q = 0x9e3779b9;
#define ROTL(x,y) (((x)<<(y&(w-1))) | ((x)>>(w-(y&(w-1)))))
#define ROTR(x,y) (((x)>>(y&(w-1))) | ((x)<<(w-(y&(w-1)))))

void RC5_ENCRYPT(WORD *pt, WORD *ct) /* 2 WORD input pt/output ct   */
{ WORD i, A=pt[0]+S[0], B=pt[1]+S[1];
   for (i=1; i<=r; i++)
      { A = ROTL(A^B,B)+S[2*i];
        B = ROTL(B^A,A)
        +S[2*i+1];
      }
   ct[0] = A; ct[1] = B;
}
void RC5_DECRYPT(WORD *ct, WORD *pt) /* 2 WORD input ct/output pt     */
{ WORD i, B=ct[1], A=ct[0];
   for (i=r; i>0; i--)
     { B = ROTR(B-S[2*i+1],A)^A;
       A = ROTR(A-S[2*i],B)^B;
      }
   pt[1] = B-S[1]; pt[0] = A-S[0];
}

void RC5_SETUP(unsigned char *K) /* secret input key K[0...b-1]               */
{   WORD i, j, k, u=w/8, A, B, L[c];
    /* Initialize L, then S, then mix key into S */
    for (i=b-1,L[c-1]=0; i!=-1; i--) L[i/u] = (L[i/u]<<8)+K[i]; for
    (S[0]=P,i=1; i<t; i++) S[i] = S[i-1]+Q;
    for (A=B=i=j=k=0; k<3*t; k++,i=(i+1)%t,j=(j+1)%c)       /* 3*t > 3*c */
       { A = S[i] = ROTL(S[i]+(A+B),3);
         B = L[j] = ROTL(L[j]+(A+B),(A+B));
       }
}

void main()
{
      WORD i,j,pt1[2],pt2[2], ct[2]={0,0};
      unsigned char key[b];
      if (sizeof(WORD)!=4)
            printf("RC5 error: WORD has %d bytes.\n",sizeof(WORD));
      printf("RC5-32/12/16 encryption and decryption:\n"); #pragma
      omp parallel for num_threads(3)
```

```
    {
        #pragma omp for
        {
            for(i=0;i<3;i++)
            {
                if(i==0)
                { pt1[0]="Hello";pt1[1]="Parag";}
                if(i==1)
                { pt1[0]="Hello";pt1[1]="Manas";
                } if(i==2)
                { pt1[0]="Hello";pt1[1]="Goyal";
                } for(j=0;j<b;j++)
                    key[j]=rand()%10;
                RC5_SETUP(key);
                RC5_ENCRYPT(pt1,ct
                );
                RC5_DECRYPT(ct,pt2
                );
                printf("\n key = ");
                for(j=0;j<b;j++)
                    printf("%.2X ",key[j]);
                                        printf("\nEncryption\n plaintext %s %s ---
> ciphertext %.8X %.8X \n",pt1[0],pt1[1],ct[0],ct[1]);
                                    printf("\nDecryption\n ciphertext %.8X %.8X ---
> plaintext %s %s \n",ct[0],ct[1],pt2[0],pt2[1]);
                if(pt1[0]!=pt2[0] || pt1[1]!=pt2[1])
                    printf("Decryption Error!");
            }
        }
    }
}
```

8. Execution Snapshots



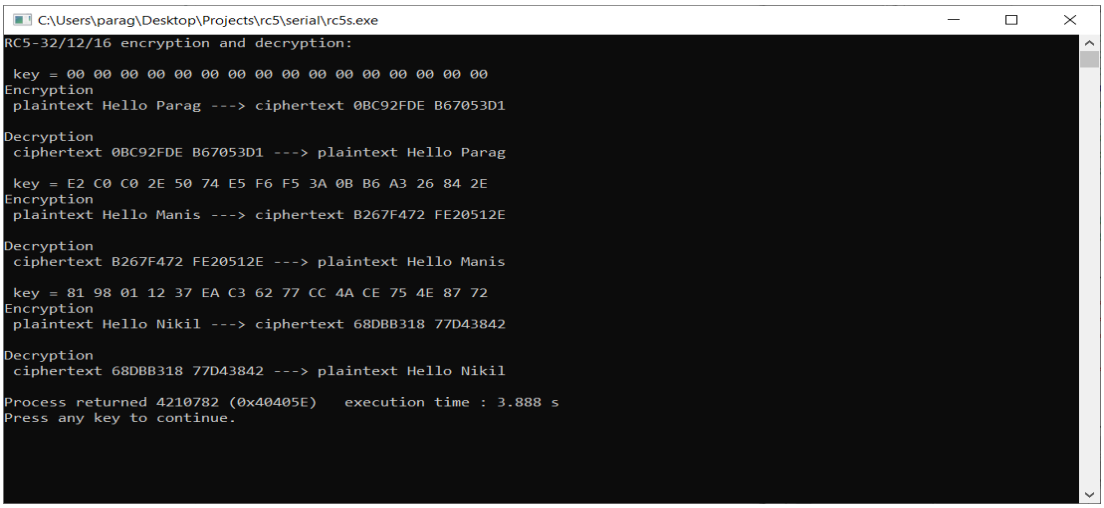
IDEA Output

```
encrypted text :
49561226755275426911182210159024535048594859233231457701590223323208242082423323145771632601208241457705275415344101457722675272101822118221233231457704956136442233230208245275419436485923323122675029154272102691129154

decrypted text : trump killed kennedy and usa dropped the nuclear bomb

time elapsed  72.50345849990845
```

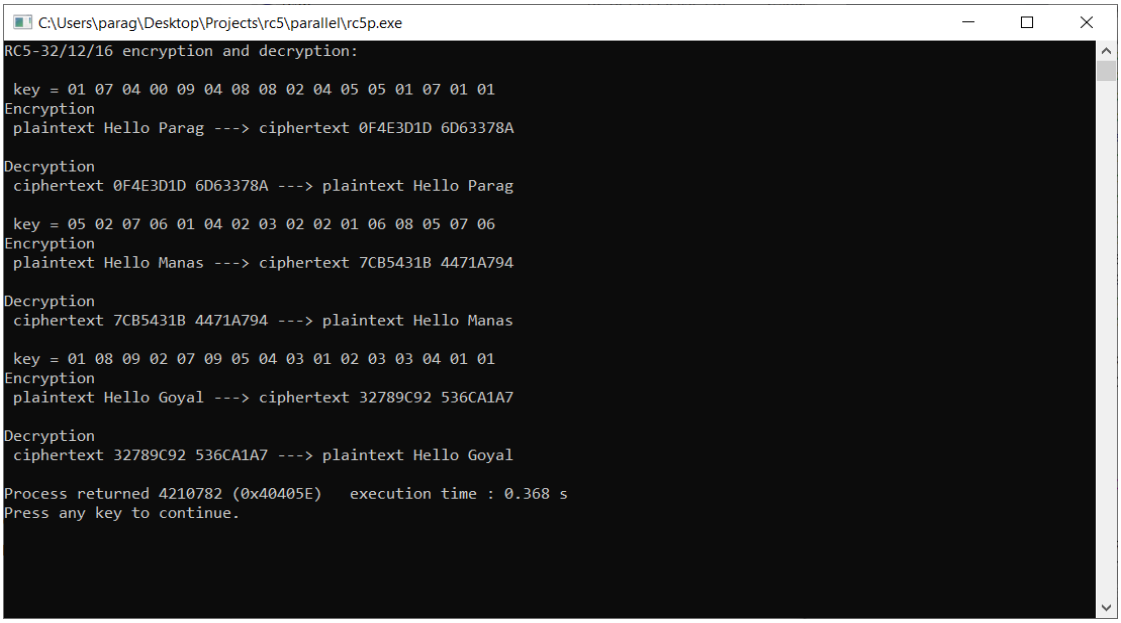
RSA Serial

```
encrypted text :
49561226755275426911182210159024535048594859233231457701590223323208242082423323145771632601208241457705275415344101457722675272101822118221233231457704956136442233230208245275419436485923323122675029154272102691129154
decrypted text : trump killed kennedy and usa dropped the nuclear bomb

time elapsed  19.344350576400757

In [2]:
```

RSA Parallel

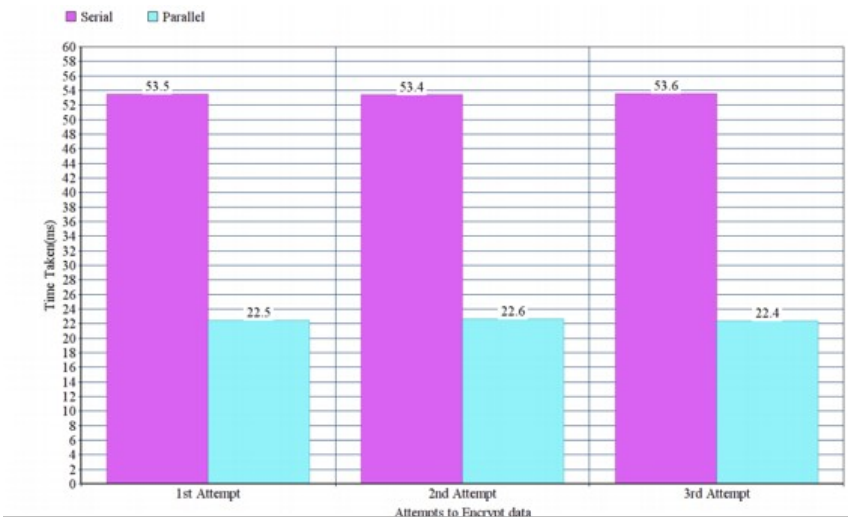


RC5 Serial

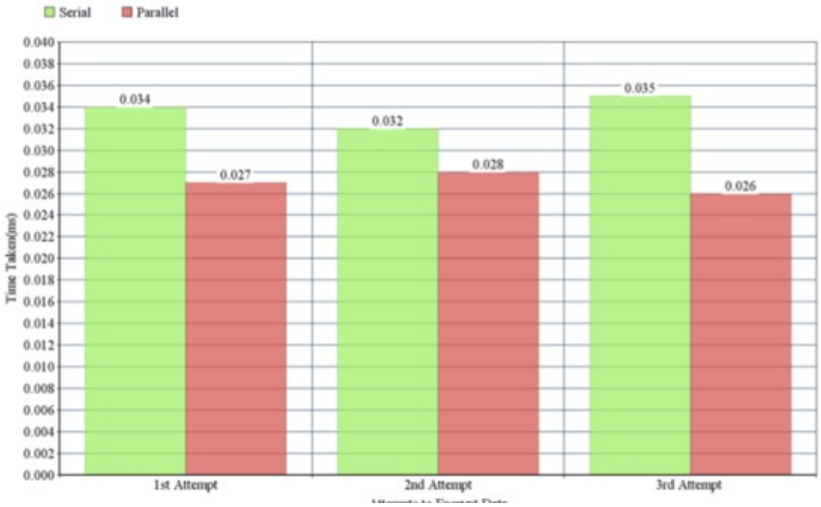


RC5 Parallel

9. Graphical comparison and Results



RSA Algorithm



RC5 Algorithm

10. Conclusion

Advantages of parallelization

- Less execution times
- Efficient code
- Distribution of instructions to threads

automated Disadvantages

- Overhead cost outweighs the time saved in case of small strings
- Memory usage multiplies by the number of threads/processors used
- Weight of performance on processor increases exponentially with the increase in number of processors

Future Study

With the hope of better parallelization algorithms and improvement in the encryption algorithms, this process can be accelerated much faster than its usual speed. Data dependency among the steps must be made a little less complicated so that they are easy to be modularised into various components. Certain algorithms must be developed that actually parallelize the core components of these algorithms.

11. Appendices

Appendix 1 : Test Cases

#1 : 59 characters
Sonos Port has a built-in digital-to-analog converter (DAC)

#2 :
In 2017, Apple introduced Face ID for the iPhone X as a replacement to Touch ID, its fingerprint technology

#3 :
Pre-orders for the Sonos Port begin today, and it'll be available starting September 12 in the U.S., with a global rollout to follow early next year.

#4 :
If you've played with the most recent smartphones from Samsung, Huawei and other Android manufacturers, you know that in-screen fingerprint readers already work quite well.

#5 :
A lot of the Port's specs are similar to the outgoing Connect's, but the more compact package, and its new matte-black look seem much better in terms of integrating unobtrusively with your existing setup.

#6 :
The asking price may seem a bit steep for what is essentially a connectivity accessory, but the Sonos Connect basically replaces a DAC entirely, which can be quite expensive on its own, and uniquely provides Sonos connectivity and streaming capabilities as well.

#7 :
The Sonos Port includes two built-in 10/100 Mbps Ethernet ports, so you can wire right into your router if you need a more reliable connection, and it has a 12V power trigger, which means it'll automatically turn on stereo or receiver equipment when they're connected and in standby mode.

#8 :
The Sonos Port includes two built-in 10/100 Mbps Ethernet ports, so you can wire right into your router if you need a more reliable connection, and it has a 12V power trigger, which means it'll automatically turn on stereo or receiver equipment when they're connected and in standby mode. Sonos Port has a built-in digital-to-analog converter (DAC)

#9 :
The Sonos Port includes two built-in 10/100 Mbps Ethernet ports, so you can wire right into your router if you need a more reliable connection, and it has a 12V power trigger, which means it'll automatically turn on stereo or receiver equipment when they're connected and in standby mode. If

you’ve played with the most recent smartphones from Samsung, Huawei and other Android manufacturers, you know that in-screen fingerprint readers already work quite well.

#10 :

The asking price may seem a bit steep for what is essentially a connectivity accessory, but the Sonos Connect basically replaces a DAC entirely, which can be quite expensive on its own, and uniquely provides Sonos connectivity and streaming capabilities as well. The Sonos Port includes two built-in 10/100 Mbps Ethernet ports, so you can wire right into your router if you need a more reliable connection, and it has a 12V power trigger, which means it’ll automatically turn on stereo or receiver equipment when they’re connected and in standby mode.

12. References

- Amalraj, A. J., & Jose, J. J. R. (2016). A survey paper on cryptography techniques. *International Journal of Computer Science and Mobile Computing*, *5*(8), 55-59.
- Mahajan, P., & Sachdeva, A. (2013). A study of encryption algorithms AES, DES and RSA for security. *Global Journal of Computer Science and Technology*.
- Venkat Prasad K., & Magesh S. (2015). A SURVEY ON ENCRYPTION ALGORITHMS USING MODERN TECHNIQUES. International Journal of Pure and Applied Mathematics
- Alemami, Y., Mohamed, M. A., & Atiewi, S. Research on Various Cryptography Techniques.
- Yang, C. C., Chang, T. S., & Jen, C. W. (1998). A new RSA cryptosystem hardware design based on Montgomery's algorithm. *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, *45*(7), 908-913.
- Huang, X., & Wang, W. (2015). A novel and efficient design for an RSA cryptosystem with a very large key size. *IEEE Transactions on Circuits and Systems II: Express Briefs*, *62*(10), 972-976.
- Chang, C. C., & Hwang, M. S. (1996). Parallel computation of the generating keys for RSA cryptosystems. *Electronics Letters*, *32*(15), 1365-1366.
- Rami Aldahdooh. Parallel Implementation and Analysis of Encryption Algorithms
- Pachori, V., Ansari, G., & Chaudhary, N. (2012). Improved performance of advance encryption standard using parallel computing. *International Journal of Engineering Research and Applications (IJERA)*, *2*(1), 967-971.
- ATTAR, N., DELDARI, H., & KALANTARI, M. (2017). AES ENCRYPTION ALGORITHM PARALLELIZATION IN ORDER TO USE BIG DATA CLOUDx