DES algorithm using RMI with JVM and OpenMP

A PROJECT REPORT

Submitted by

Suyash Pasari (16BCE0624)

Siddharth Shailendra(16BCB0129)

Aditya Firoda (16BCE2184)

Course Code: CSE4001

Course Title: Parallel and Distributed Computing

In partial fulfilment for the award of the degree of

B.Tech

in

Computer Science and Engineering

Under the guidance of

Dr. Narayanamoorthi

SCOPE,

VIT , Vellore.

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

NOVEMBER, 2019

INDEX

1. ABSTRACT

Cryptography is a field of study of mathematical algorithms which are related to specific aspects of information security such as confidentiality, integrity of content, authentication (entity wise), and data origin authentication. Even though most cryptography algorithms work better in hardware than software, systems implemented on hardware have significant drawbacks: they fail to give responses to flaws discovered in the algorithm implemented or to the changes in standards. Alternatively, it is feasible to implement cryptographic algorithms in software executing on multiple processors.

In this project we have implemented DES algorithm in serial and parallel using RMI and OpenMP.

RMI is the stepping stone of distributed object- oriented programming in Java. It explains how Java components can interact with one another in a multiple JVM environment. RMI provides the required tools that are needed to create distributed Java applications (objects) that can invoke methods on other Java applications which are running on other Java virtual machines, such as remote hosts across the Internet.

OpenMP is the short for Open Multi-Processing. It is used for parallelization of program on shared memory systems. This project implements DES algorithm and compares the time of execution, serially and parallelly using two platforms individually – RMI and OpenMP

2. INTRODUCTION

DES is the archetypal block cipher—an algorithm that takes a fixed-length string of plaintext bits and transforms it through a series of complicated operations into another ciphertext bitstring of the same length. In the case of DES, the block size is 64 bits. DES also uses a key to customize the transformation, so that decryption can supposedly only be performed by those who know the particular key used to encrypt. The key ostensibly consists of 64 bits; however, only 56 of these are actually used by the algorithm. Eight bits are used solely for checking parity, and are thereafter discarded. Hence the effective key length is 56 bits.

The key is nominally stored or transmitted as 8 bytes, each with odd parity.

The algorithm's overall structure is shown in Figure 1: there are 16 identical stages of processing, termed *rounds*. There is also an initial and final permutation, termed *IP* and *FP*, which are inverses (IP "undoes" the action of FP, and vice versa). IP and FP have no cryptographic significance, but were included in order to facilitate loading blocks in and out of mid-1970s 8-bit based hardware.

Before the main rounds, the block is divided into two 32-bit halves and processed alternately; this criss-crossing is known as the Feistal Scheme. The Feistel structure ensures that decryption and encryption are very similar processes—the only difference is that the subkeys are applied in the reverse order when decrypting. The rest of the algorithm is identical. This greatly simplifies implementation, particularly in hardware, as there is no need for separate encryption and decryption algorithms.

The $\oplus$ symbol denotes the exclusive-OR (XOR) operation. The *F-function* scrambles half a block together with some of the key. The output from the F-function is then combined with the other half of the block, and the halves are swapped before the next round. After the final round, the halves are swapped; this is a feature of the Feistel structure which makes encryption and decryption similar processes.

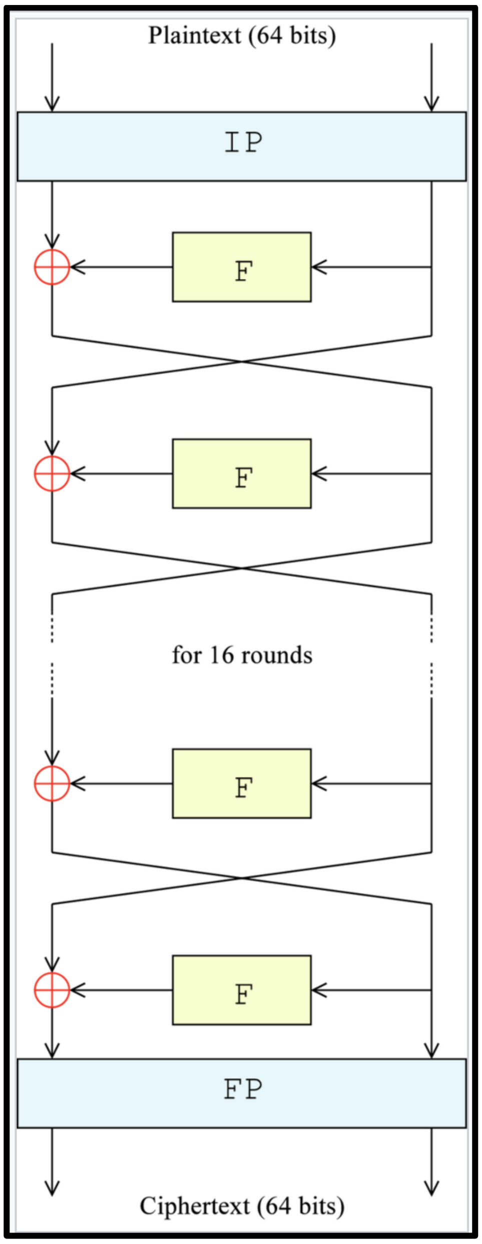


Figure 1: The overall Feistel structure of DES

The Feistel (F) function

The F-function, depicted in Figure 2, operates on half a block (32 bits) at a time and consists of four stages:

1. *Expansion*: the 32-bit half-block is expanded to 48 bits using the *expansion permutation*, denoted *E* in the diagram, by duplicating half of the bits. The output consists of eight 6-bit (8 * 6 = 48 bits) pieces, each containing a copy of 4 corresponding input bits, plus a copy of the immediately adjacent bit from each of the input pieces to either side.
2. *Key mixing*: the result is combined with a *subkey* using an XOR operation. Sixteen 48-bit subkeys—one for each round—are derived from the main key using the *key schedule* (described below).

3. *Substitution*: after mixing in the subkey, the block is divided into eight 6-bit pieces before processing by the *S-boxes*, or *substitution boxes*. Each of the eight S-boxes replaces its six input bits with four output bits according to a non-linear transformation, provided in the form of a lookup table. The S-boxes provide the core of the security of DES—without them, the cipher would be linear, and trivially breakable.
4. *Permutation*: finally, the 32 outputs from the S-boxes are rearranged according to a fixed permutation, the *P-box*. This is designed so that, after permutation, the bits from the output of each S-box in this round are spread across four different S-boxes in the next round.

The alternation of substitution from the S-boxes, and permutation of bits from the P-box and E-expansion provides so-called "confusion and diffusion" respectively, a concept identified by Claude Shannon in the 1940s as a necessary condition for a secure yet practical cipher.

3. LITERATURE SURVEY

PAPER NAME	KEYWORDS	CONCLUSION	YEAR
Improved Performance of Advance Encryption Standard using Parallel Computing	AES, Data Parallelism Control Parallelism, Parallel computing	• Both the proposed approach of AES algorithm is successfully implemented in JPPF framework and performance is measured in terms of execution time. [1] • It can be seen that the experiment shows a significant improvement in terms of execution time • AES is implemented using JPPF framework. It can also be implemented using other fast framework available • The network traffic can affect the performance of this implementation	2012
Parallelization of the Data Encryption Standard (DES) algorithm	cryptography, Data Encryption Standard, OpenMP API, data dependence analysis	• Using the memory with more latency (for example, using the interleaving technique) we can even increase the speed-up of the whole parallel program • The hardware synthesis of the DES algorithm will depend on appropriate adjustment of the data transmission capacity and the computational power of hardware. • the code containing I/O functions is unparallelizable, because the access to memory is, by its very nature, sequential • the total speed-up received on PC computer is about 1.86. [2]	2011

PAPER NAME	KEYWORDS	CONCLUSION	YEAR
Task Parallelism using Distributed Computing for Encryption and Decryption	Distributed computing, parallel processing, data conversion, distributed programming, Advanced Encryption Standard Algorithm.	• This report presents the most efficient, currently known approaches in encryption and decryption of text with AES on parallel processing to achieve great speed. [3] • If the amount of data is large, the encryption/decryption time required is greatly reduced, if it runs on a parallel processing in a distributed environment. • Parallel implementations of hashing and public key algorithms may also be implemented, in order to create a complete cryptographic framework in a distributed environment.	2014
A Study of Encryption Algorithms (RSA, DES, 3DES and AES) for Information Security	Encryption, RSA, DES, 3DES, AES	• all the techniques are useful for real-time Encryption • AES algorithm is most efficient in terms of speed, time, throughput and avalanche effect. [4] • The Security provided by these algorithms can be enhanced further, if more than one algorithm is applied to data.	2013
Data Encryption Standard Algorithm (DES) for Secure Data Transmission	Cryptography, symmetric key, asymmetric key, DES algorithm	• DES is now considered to be an insecure technique of encryption for some applications like banking system. [5] • There are some analytical results which demonstrate theoretical weaknesses in the cipher. So, it becomes very important to augment this algorithm by adding new level of security to it.	2016

PAPER NAME	KEYWORDS	CONCLUSION	YEAR
Modification Of Des Algorithm	wireless communication, security, pseudorandom generator, advance cryptography	• Customized cryptography ensures user authentication, bandwidth sharing, and security from eavesdropping and immunity to interference, and difficulty in detection. • Thus, advance cryptography is better than other cryptographic techniques because it is efficient, fast and reliable technique. [6]	2012
A Modified Simplified Data Encryption Standard Algorithm	S-DES, Encryption, Decryption, Cryptography, Random Number, Security.	• When we communicate with another person we threat regarding security in paper we find out solution to make more secure S-DES algorithm. [7] • In this paper we propose random key generation to make attacker difficult to guess.	2017
Comparative Analysis of AES and DES security Algorithms	AES, DES, Cryptography, Symmetric key, Asymmetric key, Encryption, Decryption	• In cryptography we use key called public key and private key. With the help of these keys we encrypt and decrypt the data from make secure. Encrypted data is called ciphertext and decrypted data s called plaintext. • Symmetric key algorithms are those algorithms which have same key for encryption and decryption. [8]	2013
Implementation Cryptography Data Encryption Standard(DES) and Triple Data Encryption Standard (3DES) Method in Communication System Based Near Field Communication(NFC)	Cryptography, Data Encryption Standard (DES), Triple Data Encryption Standard (3DES), NFC	• DES and 3DES text data cryptographic method can be implemented on application of ACOS3 smart card data writing process and data reading process in NFC - based systems. [9] • The execution time of ACOS3 smart card data writing process and data reading process using DES cryptographic method is faster than 3DES	2009

PAPER NAME	KEYWORDS	CONCLUSION	YEAR
		cryptographic method. [9] • The execution time of data reading process is faster than data writing process for ACOS3 smart card, using DES or 3DES cryptographic method.	
Design and Simulation DES Algorithm of Encryption for Information Security	encryption, C##,DES, Decryption, Cryptography, file, cipher text.	• In cryptography, encryption and decryption algorithms runs important role in network security. • algorithm consumes least encryption time and DES consume maximum encryption time. [10]	2018
A Survey on Performance Analysis of DES, AES and RSA Algorithm along with LSB Substitution Technique	Cryptography, Steganography, DES, RSA, AES, LSB.	• These encryption techniques are studied and analyzed well to promote the performance of the encryption methods also to ensure the security. • Based on the experimental result it was concluded that AES algorithm consumes least encryption and decryption time and buffer usage compared to DES algorithm. [11] • RSA consume more encryption time and buffer usage is also very high. [11] • we observed that decryption of AES algorithm is better than other algorithms. • From the simulation result, we evaluated that AES algorithm is much better than DES and RSA algorithm.	2013

PAPER NAME	KEYWORDS	CONCLUSION	YEAR
A Research paper: An ASCII value based data encryption algorithm and its comparison with other symmetric data encryption algorithms	Encryption, Decryption, ASCII, symmetric encryption, plaintext, ciphertext	• The proposed algorithm has the following limitations: 1) More Execution time 2) Key Length and length of plain text must be same. [12] • In the future work related to proposed algorithm, the limitations of proposed algorithm are overcome by encrypting and decrypting data with may or may not be same key length size in comparison with input size.	2012
Future-Based RMI: Optimizing Compositions of Remote Method Calls on the Grid	Exception Handling, Method Invocation, Remote Method	• In this optimized RMI, methods are not invoked directly. Instead they work on remote reference to the objects. [13] • Pool of threads is created to avoid thread-creation overhead.	2003
Object Serialization and Remote Method Invocation	Client Application, Method Invocation, Client Program, Remote Procedure, Call	• State of a Java object is preserved so that it can be stored and used to restore to its original state later, if required. [14]	2008
Java and Remote Method Invocation	Remote Server, Remote Object	• RMI is preferable to sockets when Java to Java communication comes into picture. [15]	2003

4. OVERVIEW OF THE WORK

4.1. PROBLEM DESCRIPTION

Security measures must be incorporated into computer systems whenever they are potential targets for malicious or mischievous attacks. This is especially for systems which handle financial transactions or confidential, classified or other information whose secrecy and integrity are critical. With the need to protect the integrity and privacy of information belonging to individuals and organizations, we have developed this system.

The objective of encryption algorithms is to allow the encryption of a message by one person and the decryption of the message by another person. This project aims at converting the plaintext into a form unreadable by unauthorized people and hence can be readily transferred across the web and decrypted at the recipient side only by authorized people. It provides an interactive environment to encrypt, decrypt or transfer encrypted files without compromising with the integrity and privacy of critical information. In the era of wide area, open distributed systems , this system will help resolve various security issues.

4.2. HARDWARE REQUIREMENTS

We are making a project on security for which we have used Java RMI and the system requirement are as follows:

- Intel Core i3
- RAM Size – 256 MB
- Hard Disk – 1 GB

4.3. SOFTWARE REQUIREMENTS

- MacOS El Capitan (or later versions).
- JDK 1.6.0

5. SYSTEM DESIGN

5.1. MODULE DESCRIPTION

There are 4 modules in our project

- GUI (Graphical User Interface) - GUI provides an efficient way through which user can interact with the system and works accordingly to fulfill the task. User can generate keys randomly or enter manually. Thus it provides a user friendly environment.
- Key Generation - This module generates keys. The keys can be generated randomly or it can be specified by the user.
- Encryption – Encryption is a primary method of protecting valuable electronic information. Encryption is the process of encoding a message in such a way as to hide it"s contents.
- Decryption - the process of restoring the plain text from the cipher text is known as decryption.

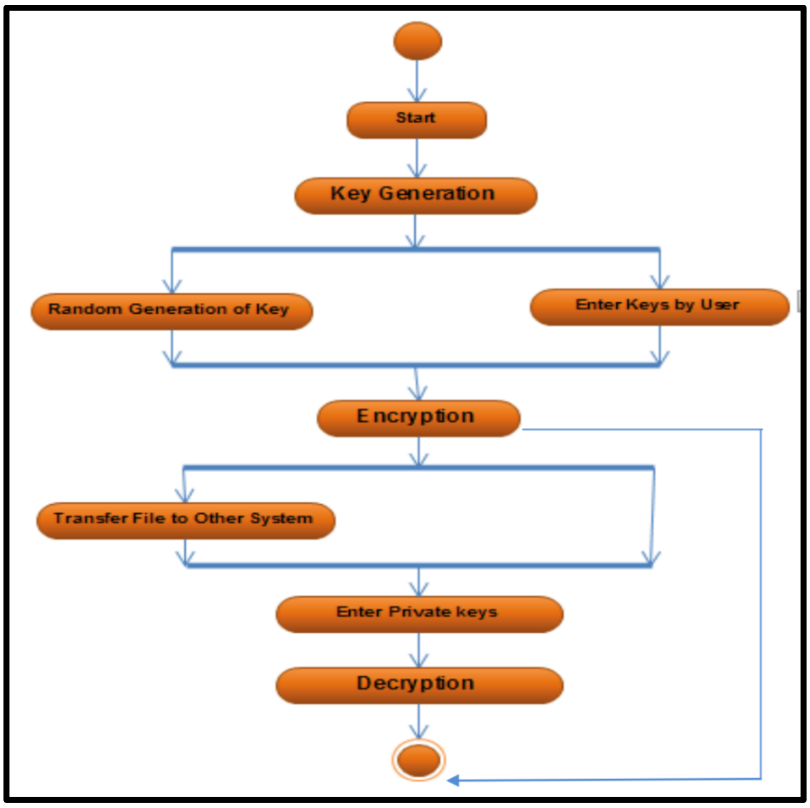


Figure 2: Activity Diagram

5.2 HOW TO RUN

- Compile All Files

  javac *.java

- Create stub and skeleton using the rmic tool

  rmic DecryptorRemote

- Start RMI in one terminal

  rmiregistry 5000

- Start server in Server Node

  java RMIServer

- Start RMI Client in Client Node

  java RMIClient

6. IMPLEMENTATION

RMIServer.java

```
import java.rmi.*;
import java.rmi.registry.*;
public class RMIServer{
    public static void main(String args[]){
        try{
            Decryptor stub = new DecryptorRemote();
            Naming.rebind("rmi://localhost:5000/parallel", stub);
        }catch(Exception e){e.printStackTrace();}
    }
}
```

RMIClient.java

```
import java.rmi.*;
public class RMIClient{
    public static void main(String args[]){
        try{
            Decryptor stub=(Decryptor)Naming.lookup("rmi://localhost:5000/parallel");
            String x = new String("This is a test");
            byte [] encrypted = stub.encrypt(x.getBytes());
            byte [] decrypted = stub.decrypt(encrypted);
        }catch(Exception e){}
    }
}
```

RMIApplet.java

```
import java.rmi.*;
import java.awt.event.*;
import javax.swing.*;
public class RMIApplet{
    public static void main(String[] args)throws Exception{
        JFrame f=new JFrame("Button Example");
        final JTextField tf=new JTextField();
        tf.setBounds(125,50,150,20);
        JButton b=new JButton("Serial Execution");
        JButton b1=new JButton("Parallel Execution");
        b.setBounds(50,100,150,30);
        b1.setBounds(200,100,150,30);
        b.addActionListener(new ActionListener(){
            public void actionPerformed(ActionEvent e){
                try{
                    DES des = new DES();
                    long startTime = System.nanoTime();
                    String s = "This is a test";
                    String s1 = des.bruteForce(s.getBytes());
                    System.out.println(s1);
                    long difference = System.nanoTime() - startTime;
                    tf.setText(difference / 1000000 + "ms");
                } catch(Exception e1){}
            }
        });
        b1.addActionListener(new ActionListener(){
            public void actionPerformed(ActionEvent e){
                try{
                    long startTime = System.nanoTime();
                    Decryptor stub=(Decryptor)Naming.lookup("rmi://localhost:5000/parallel");
                    String x = new String("This is a test");
                    byte [] encrypted = stub.encrypt(x.getBytes());
                    byte [] decrypted = stub.decrypt(encrypted);
                    long difference = System.nanoTime() - startTime;
                    tf.setText(difference / 1000000 + "ms");
                } catch(Exception e1){}
              }
          });
          f.add(b);f.add(tf);f.add(b1);
          f.setSize(400,400);
          f.setLayout(null);
          f.setVisible(true);
      }
  }
```

DES.java

```
/*The Data Encryption Standard is a symmetric-key algorithm for the encryption of electronic
data.
 Although now considered insecure, it was highly influential in the advancement of modern
cryptography.
 an algorithm that takes a fixed-length string of plaintext bits.
transforms it through a series of complicated operations into another ciphertext bitstring of the
same length. */
import java.security.SecureRandom;
import javax.crypto.Cipher;
import javax.crypto.KeyGenerator;
import javax.crypto.SecretKey;
import javax.crypto.spec.SecretKeySpec;
import java.util.Random;
import java.util.Scanner;
class DES {
   byte[] skey = new byte[1000];
   String skeyString;
```

```
    static byte[] raw;
    String inputMessage,encryptedData,decryptedMessage;
    Scanner sca;
    public DES() {
        try {
            // sca = new Scanner(System.in);
            generateSymmetricKey();
            // System.out.println("Enter message to encrypt");
            // inputMessage=sca.nextLine(); //enter the message for encryption
            // byte[] ibyte = inputMessage.getBytes(); //extract bytes from the input message
            // byte[] ebyte=encrypt(raw, ibyte); //function call to encrypt
            // String encryptedData = new String(ebyte);
            // System.out.println("Encrypted message "+encryptedData);
            // byte[] dbyte= decrypt(raw,ebyte); //function call to decrypt
            // String decryptedMessage = new String(dbyte);
            //   System.out.println("Decrypted   message   "+decryptedMessage);   //display   the
decrypted message
        }
        catch(Exception e) { //exception catching
            e.printStackTrace();
        }
    }
    void generateSymmetricKey() { //function to generate a symmetric key
        try {
            Random r = new Random(); //generates a random number
            int num = r.nextInt(10000); //generates the next random number
            String knum = String.valueOf(num); //extracts the value of the string
            byte[] knumb = knum.getBytes(); //extract the bytes
            skey=getRawKey(knumb);
            skeyString = new String(skey);
            System.out.println("DES Symmetric key = "+skeyString);
        }
        catch(Exception e) {
            e.printStackTrace();
        }
    }
    public static byte[] getRawKey(byte[] seed) throws Exception {
        KeyGenerator kgen = KeyGenerator.getInstance("DES"); //secret (symmetric) key
generator constructed using one of the getInstance class methods of this class
        SecureRandom sr = SecureRandom.getInstance("SHA1PRNG");//Returns a
SecureRandom object that implements the specified Random Number Generator (RNG)
algorithm.
        sr.setSeed(seed); //Reseeds the random object
        kgen.init(56, sr); //initializes the keygenerator
        SecretKey skey = kgen.generateKey();
        raw = skey.getEncoded();
        return raw;
    }
    public static byte[] encrypt(byte[] raw, byte[] clear) throws Exception {
            SecretKeySpec skeySpec = new SecretKeySpec(raw, "DES");//is useful for raw secret
 keys that can be represented as a byte array and have no key parameters associated with them
                        Cipher cipher = Cipher.getInstance("DES");// passes the name of the requested
transformation to it
        cipher.init(Cipher.ENCRYPT_MODE, skeySpec); //resets this cipher object to the state
it was in when previously initialized via a call to init
        byte[] encrypted = cipher.doFinal(clear);
        return encrypted;
    }
    public static byte[] decrypt(byte[] raw, byte[] encrypted) throws Exception { //function to
decrypt the cipher
            SecretKeySpec skeySpec = new SecretKeySpec(raw, "DES"); //is useful for raw secret
 keys that can be represented as a byte array and have no key parameters associated with them
                        Cipher cipher = Cipher.getInstance("DES");// passes the name of the requested
transformation to it
        cipher.init(Cipher.DECRYPT_MODE, skeySpec);//resets this cipher object to the state it
was in when previously initialized via a call to init
```

```
        byte[] decrypted = cipher.doFinal(encrypted);
        return decrypted;
    }
    public static String bruteForce(byte []m) throws Exception{
        byte []e = encrypt(raw, m);
        for(int i = 0;i < 10000; i++){
            String knum = String.valueOf(i); //extracts the value of the string
            byte[] knumb = knum.getBytes(); //extract the bytes
            byte []skey=getRawKey(knumb);
            String skeyString = new String(skey);
            System.out.println("DES Symmetric key = "+skeyString);
            if(new String(encrypt(skey, e)).equals(new String(m)))
                return "" + i;
        }
        return "";
    }
    public static void main(String args[]) {
        DES des = new DES();
    }
}
```

DecryptorRemote.java

```
import java.rmi.*;
import java.rmi.server.*;
public class DecryptorRemote extends UnicastRemoteObject implements Decryptor{
    DES des;
    byte [] raw;
    DecryptorRemote()throws RemoteException{
        super();
        des = new DES();
        raw = des.raw;
    }
    public byte[] encrypt(byte [] b) throws RemoteException{
        System.out.println(des);
        byte[] encryptedString = null;
        try{
            encryptedString = des.encrypt(raw, b);
            System.out.println("Encrypted String: " + new String(encryptedString));
        } catch(Exception e){
            e.printStackTrace();
        } finally{
            return encryptedString;
        }
    }
    public byte[] decrypt(byte [] b)throws RemoteException{
        byte[] decryptedString = null;
        try{
            decryptedString = des.decrypt(raw, b);
            System.out.println("Decrypted String: " + new String(decryptedString));
        } catch(Exception e){
            e.printStackTrace();
        } finally{
            return decryptedString;
        }
    }
}
```

Decryptor.java

```
import java.rmi.*;
public interface Decryptor extends Remote{
    public byte[] decrypt(byte[] m)throws RemoteException;
    public byte[] encrypt(byte[] m)throws RemoteException;
}
```

IMPLEMENTATION (OpenMP)

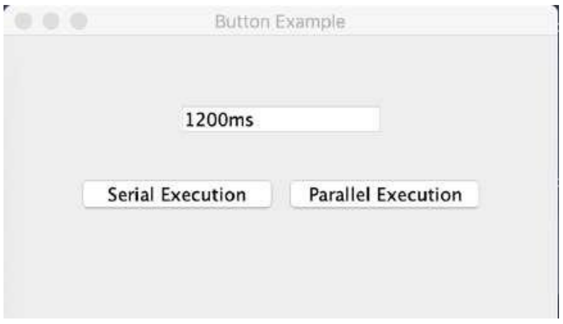
Main.cpp

```
/*-************************
* Compiler errors
*************************/
#ifndef _OPENMP
# error "OPENMP framework is required"
#endif


/*-************************
* Dependencies
*************************/
#include <omp.h>
#include <chrono>
#include <math.h>
#include <fstream>
#include <algorithm>
#include "ArgsParser.h"
#include "crypt3.h"

/*-************************
* Constants
*************************/

#define BYTE_PSW_CRYPTED 16
#define SALT_LENGTH 2
#define N_SYMBOLS 10 //The number of accepted symbols for the bruteforce ([0...9])


/*-************************
* Functions declarations
*************************/


double dictionary_attack(char* psw, char * salt, const std::string& dict_path, const int&
n_threads, const bool& random);
double bruteForce_attack(char *psw, char *salt, const int& n_threads);
inline int myrandom (int i) { return std::rand()%i;}
void getStringDictionary(std::vector<std::string>& dictionary, const std::string& dict_path,
const bool& random);
void generateCombination(int v[], int k, int i);




/**
 * Given an index i generate a k-characters combination and save it in v[] array
 */
```

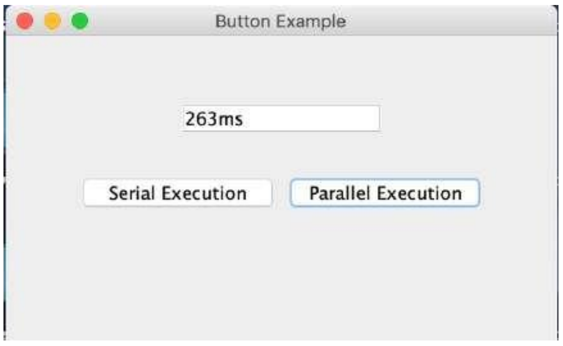
```
void generateCombination(int v[], int k, int i){
   for (int j=0; j<(k-1);j++){
      v[j] = i / (int)(pow((double) N_SYMBOLS ,(double)((k-1)-j)));
      i = i % (int)(pow((double) N_SYMBOLS ,(double)((k-1)-j)));
   }
   v[k-1] = i % N_SYMBOLS;
}

/**
 * Get the dictionary located in dict_path randomizing final items if required
 * @param dictionary data structure that will store the resulting dictionary
 * @param dict_path path where is currently stored the dictionary file
 * @param random indicates if the resulting dictionary will be randomized
 */
void getStringDictionary(std::vector<std::string>& dictionary, const std::string& dict_path,
const bool& random){
   std::ifstream dict;
   dict.open(dict_path);
   std::string line;
   if(dict.is_open()){

      while ( getline (dict,line) )
      {
         dictionary.push_back(line);
      }
      dict.close();
   }
   if (random) {
      srand(time(NULL));
      std::random_shuffle(dictionary.begin(), dictionary.end(), myrandom);
   }
}


/**
 * Performs a dictionary attack to find the password passed by argument
 * @param psw the password
 * @param salt the salt
 * @param dict_path path where is currently stored the dictionary file
 * @param n_threads the number of threads executing the key-search loop
 * @param random a bool variable indicates if randomize the dictionary loaded
 * @return the time passed to find the password
 */
double dictionary_attack(char* psw, char * salt, const std::string& dict_path, const int&
n_threads, const bool& random){

   std::vector<std::string> dictionary;
   getStringDictionary(dictionary, dict_path, random);
   char crypted[BYTE_PSW_CRYPTED];
   crypt(psw,salt,crypted);
```

```
    printf("Password crypted: %s", crypted);
    printf("\n\nSearching...\n\n");

    std::chrono::duration<double> timePassed;
    auto start = std::chrono::high_resolution_clock::now();

# pragma omp parallel num_threads(n_threads)
    {
        char* comb = (char*)malloc(8);
        char guess[BYTE_PSW_CRYPTED];

#       pragma omp for schedule(static)
        for(int i = 0;i<dictionary.size(); i++) {
            strcpy(comb, dictionary.at(i).c_str());
            //printf("\n%s",comb); //uncomment this line to view all combination tested

            crypt(comb, salt,guess);

            if (strcmp(guess,crypted)==0) {
                auto end = std::chrono::high_resolution_clock::now();
                timePassed = end - start;
                printf("\nPassword found: %s", comb);
                printf("\nComputation time: %f s",timePassed.count());
            }
        }

        free(comb);
    }
    return timePassed.count();
}



/**
 *  Performs a brute force attack to find the password passed by argument
 * @param psw the password
 * @param salt the salt
 * @param n_threads the number of threads executing the key-search loop
 * @return the time passed to find the password
 */
double bruteForce_attack(char *psw, char *salt, const int& n_threads) {

    int k = strlen(psw) + strlen(salt);
    char symbols[] = "0123456789";
    int n = strlen(symbols);
    char crypted[BYTE_PSW_CRYPTED];
    long size = (long)pow((double)n,(double)k);
    crypt(psw,salt,crypted);
    printf("Password crypted: %s", crypted);
    printf("\n\nSearching...\n");

    std::chrono::duration<double> timePassed;
    auto start = std::chrono::steady_clock::now();

# pragma omp parallel num_threads(n_threads)
    {
        int test[k];
        char* comb = (char*)malloc(k);
        char* attack_psw = (char*)malloc(strlen(psw));
        char attack_salt[SALT_LENGTH];
        char guess[BYTE_PSW_CRYPTED];

#       pragma omp for schedule(static)
        for(int i=0; i<size; i++){

            generateCombination(test, k, i);

            //Convert test to char array
            for (int j =0; j<k; j++){
                sprintf(&(comb[j]), "%d", test[j]);
            }

            strncpy(attack_psw,comb,strlen(psw)); //copy only the password
            attack_salt[0] = comb[strlen(comb)-strlen(salt)]; //take first salt digit
            attack_salt[1] = comb[strlen(comb)-strlen(salt)+1]; //take second salt digit

            //uncomment under line to view all combination tested
            //printf("\nThread %d test the string %s", omp_get_thread_num(), comb);

            crypt(attack_psw, attack_salt,guess);

            if (strcmp(guess,crypted)==0) {
                auto end = std::chrono::steady_clock::now();
                timePassed = end - start;
                printf("\nPassword found: %s", attack_psw);
                printf("\nComputation time: %f s",timePassed.count());
            }
        }
        free(comb);
        free(attack_psw);
    }
    return timePassed.count();
}



int main(int argc, char* argv[]) {
    CLParser parser(argc,argv, true);
    char psw[strlen(parser.get_arg(1).c_str())];
    char salt[2];
    int n_threads = omp_get_num_procs();
    std::string path;
    double time_passed;

    if (argc < 3 || parser.get_arg(1)=="--help" || argc> 8){

        printf("\nUSAGE:  ");
        printf("ompDES_cracker    <8 characters password> <2 characters salt> [Options]\n\n");
        printf("OPTIONS:\n");
        printf("-d    <dictionary_path>: Enable dictionary attack (default: brute force attack)\n");
        printf("-nt    <num_threads>:        Set the number of threads (default: number of logical
cores)\n");
        printf("-r                          Randomize attempts (only for dictionary) \n\n");
        return 0;

    }

    strcpy(psw,parser.get_arg(1).c_str());
    strcpy(salt,parser.get_arg(2).c_str());

    if (parser.get_arg("-nt")!=""){
        n_threads = atoi(parser.get_arg("-nt").c_str());
    }
    if (parser.get_arg("-d")!=""){
        path = parser.get_arg("-d");
        printf("\n-----SELECTED MODE: DICTIONARY -----\n\n");
        time_passed = dictionary_attack(psw,salt,path,n_threads,parser.find_arg("-r"));

    }
    else {
        printf("\n-----SELECTED MODE: BRUTEFORCE ---- \n\n");
        time_passed = bruteForce_attack(psw,salt,n_threads);
    }


    return 0;
}
```

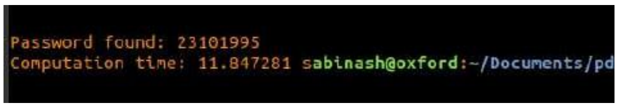
PERFORMANCE PARAMETERS:



Serial execution time in RMI = 1200ms

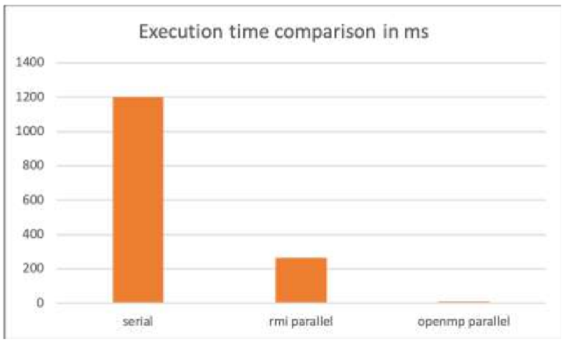


Parallel execution time in RMI= 263ms



```
Password found: 23101995
Computation time: 11.847281 sabinash@oxford:~/Documents/pd
```

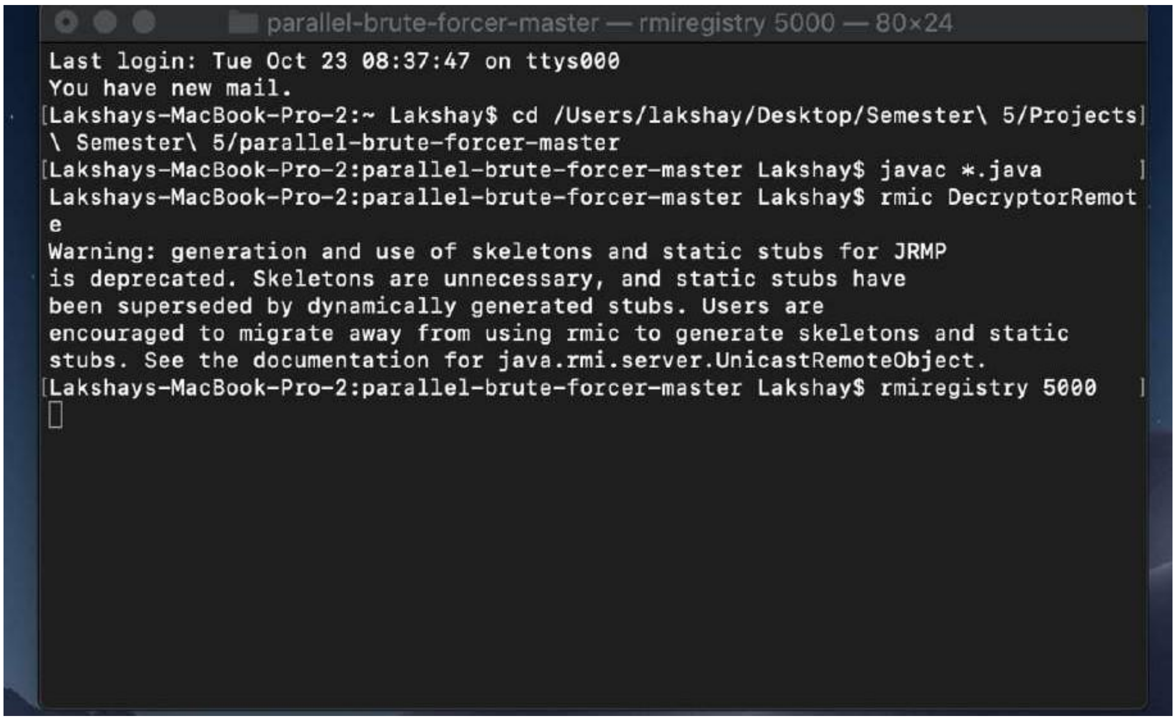
Parallel execution time in OpenMP = 11.847ms



EXISTING SYSTEM:

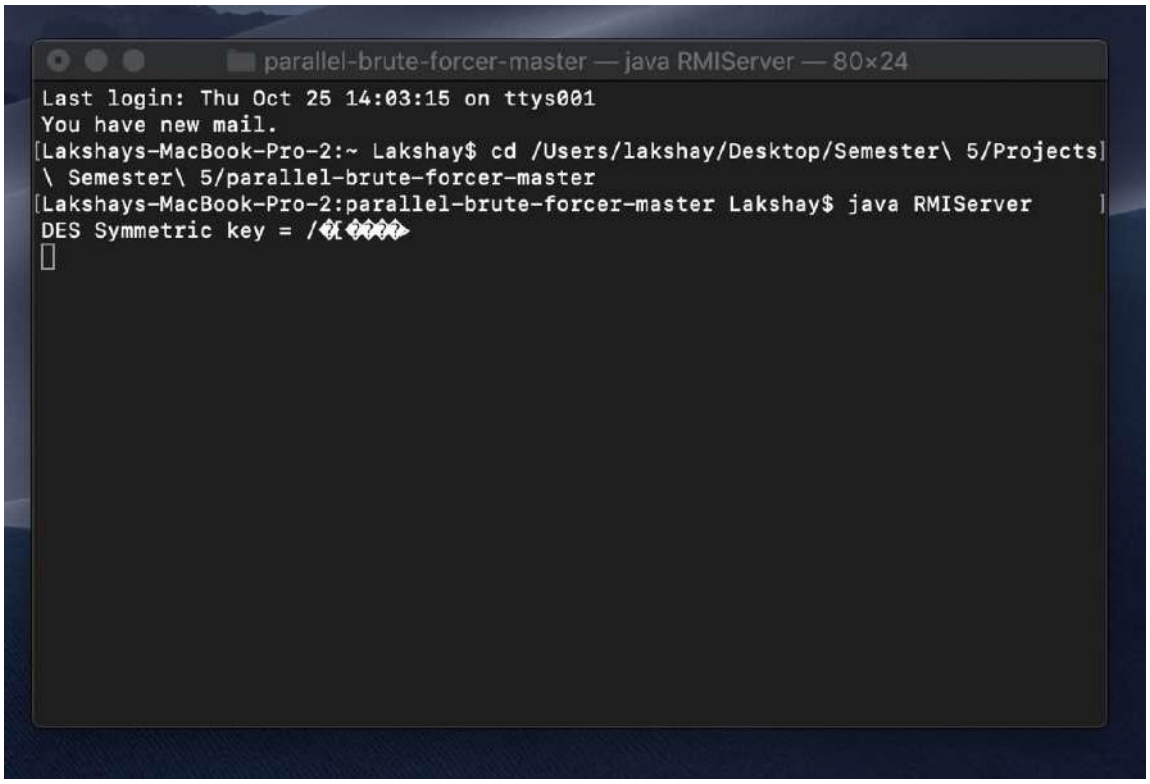
- Considering improvement in performance of advanced encryption standard using Parallel Computing, here the problem faced was that faster frameworks could also have been used for implementation. Also, this method is vulnerable to network traffic, which can affect its performance.
- While another pre-existing method is that of Parallelization of the Data Encryption Standard Algorithm. Here, memory with more latency is made use of so that there is an effective increase in the speed-up of the entire program (parallel) except the I/O function region because of its sequential nature.
- Another method talks about how if large amount of data is input then the encryption/decryption time required is greatly reduced, provided the processing is parallel execution under distributed environment.
- Yet further a study talks about how application of all standard algorithms (RSA, DES, 3DES, AES) when applied to a piece of information then the security parameter is enhanced by greater percent.
- For applications like banking system, the DES algorithm’s effectiveness is being questioned due to the result of certain analytical results which demonstrate the weakness in the cipher.
- A Modified Simplified Data Encryption Standard Algorithm, a random key generation was done so that the attacker finds it difficult to guess the key and hence the vulnerability of communication data to be transmitted is reduced.

IMPLEMENTATION RESULTS



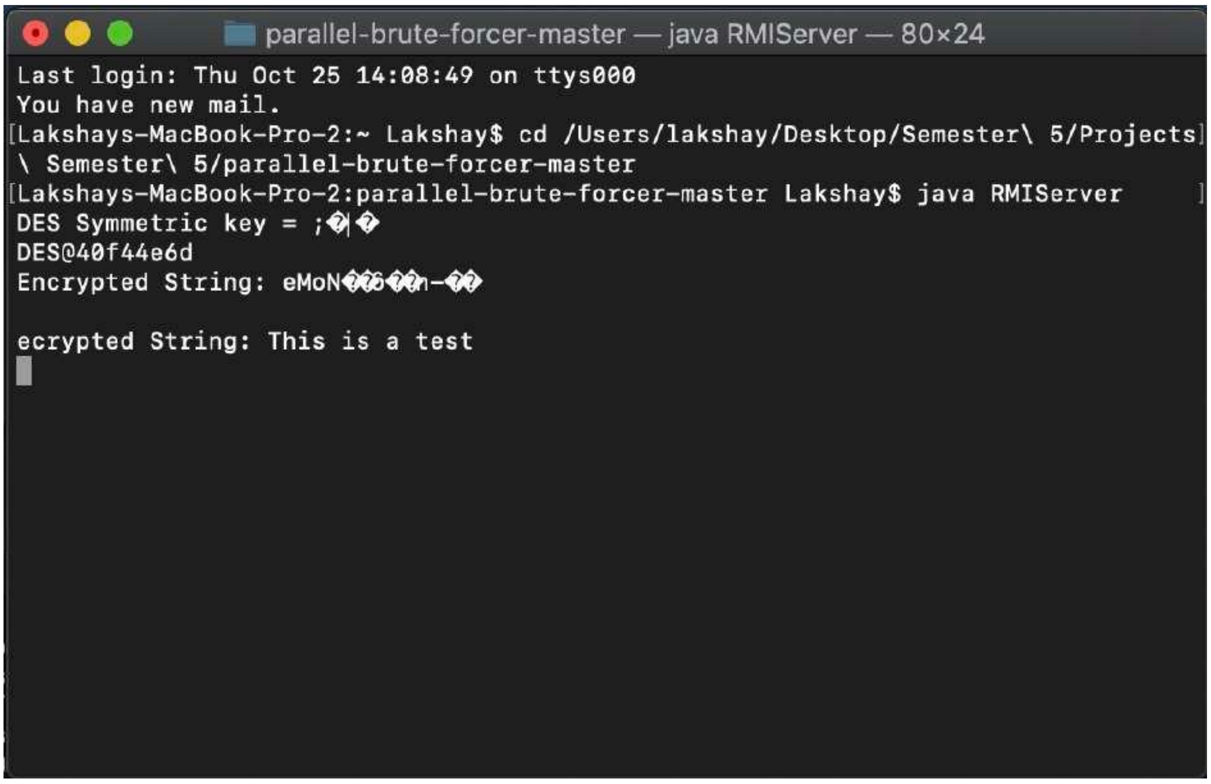
```
Last login: Tue Oct 23 08:37:47 on ttys000
You have new mail.
[Lakshays-MacBook-Pro-2:~ Lakshay$ cd /Users/lakshay/Desktop/Semester\ 5/Projects]
\ Semester\ 5/parallel-brute-forcer-master
[Lakshays-MacBook-Pro-2:parallel-brute-forcer-master Lakshay$ javac *.java       ]
Lakshays-MacBook-Pro-2:parallel-brute-forcer-master Lakshay$ rmic DecryptorRemot
e
Warning: generation and use of skeletons and static stubs for JRMP
is deprecated. Skeletons are unnecessary, and static stubs have
been superseded by dynamically generated stubs. Users are
encouraged to migrate away from using rmic to generate skeletons and static
stubs. See the documentation for java.rmi.server.UnicastRemoteObject.
[Lakshays-MacBook-Pro-2:parallel-brute-forcer-master Lakshay$ rmiregistry 5000   ]
```

Figure 1: Importing all java files and declaring registry 5000



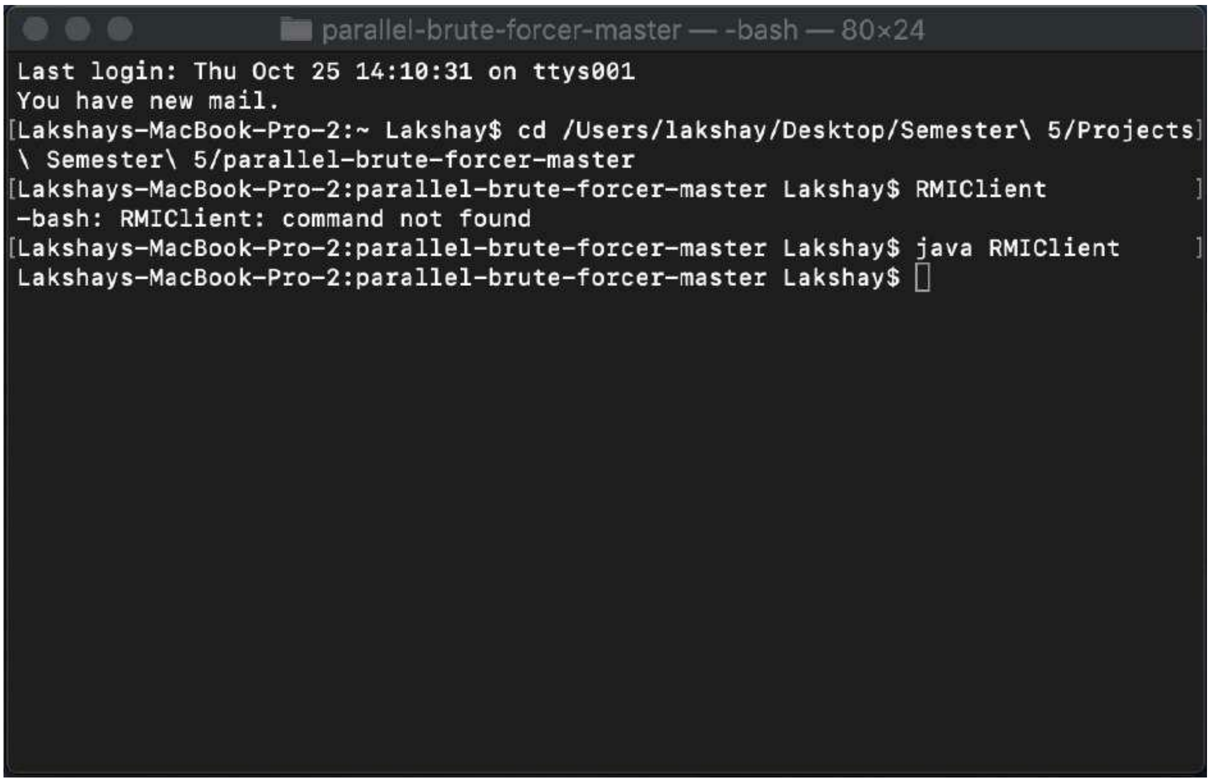
```
Last login: Thu Oct 25 14:03:15 on ttys001
You have new mail.
[Lakshays-MacBook-Pro-2:~ Lakshay$ cd /Users/lakshay/Desktop/Semester\ 5/Projects]
\ Semester\ 5/parallel-brute-forcer-master
[Lakshays-MacBook-Pro-2:parallel-brute-forcer-master Lakshay$ java RMIServer     ]
DES Symmetric key = /�ɪ����
```

Figure 2: Calling the RMIServer



```
Last login: Thu Oct 25 14:08:49 on ttys000
You have new mail.
[Lakshays-MacBook-Pro-2:~ Lakshay$ cd /Users/lakshay/Desktop/Semester\ 5/Projects]
\ Semester\ 5/parallel-brute-forcer-master
[Lakshays-MacBook-Pro-2:parallel-brute-forcer-master Lakshay$ java RMIServer     ]
DES Symmetric key = ;�|�
DES@40f44e6d
Encrypted String: eMoN��6��n-��

ecrypted String: This is a test
```

Figure 3: Show encrypted and decrypted string



```
Last login: Thu Oct 25 14:10:31 on ttys001
You have new mail.
[Lakshays-MacBook-Pro-2:~ Lakshay$ cd /Users/lakshay/Desktop/Semester\ 5/Projects]
\ Semester\ 5/parallel-brute-forcer-master
[Lakshays-MacBook-Pro-2:parallel-brute-forcer-master Lakshay$ RMIClient          ]
-bash: RMIClient: command not found
[Lakshays-MacBook-Pro-2:parallel-brute-forcer-master Lakshay$ java RMIClient     ]
Lakshays-MacBook-Pro-2:parallel-brute-forcer-master Lakshay$
```

Figure 4: Calling the RMIClient

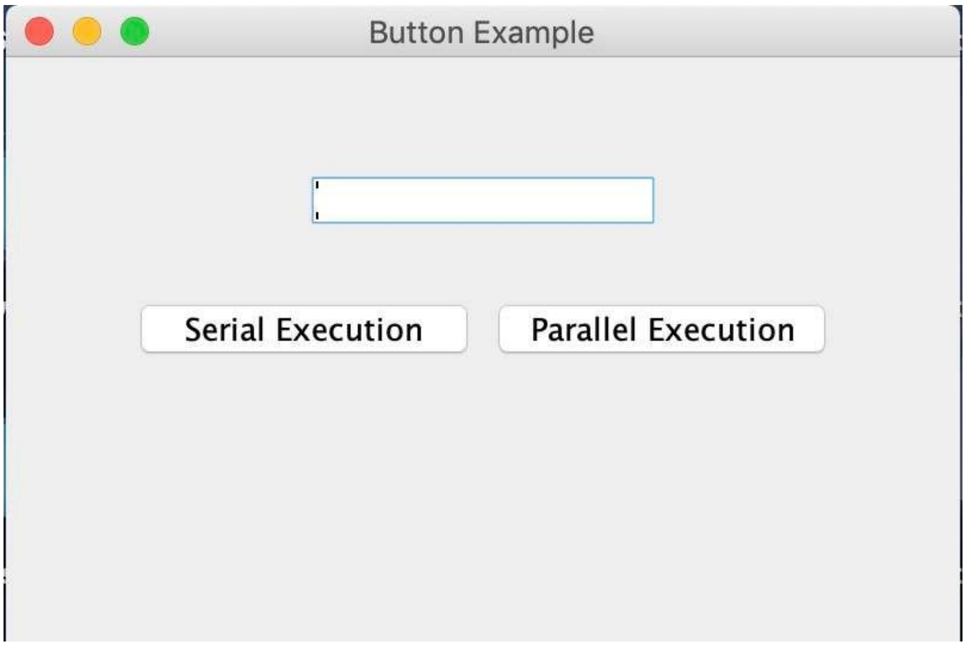


Figure 5: RMIApplet

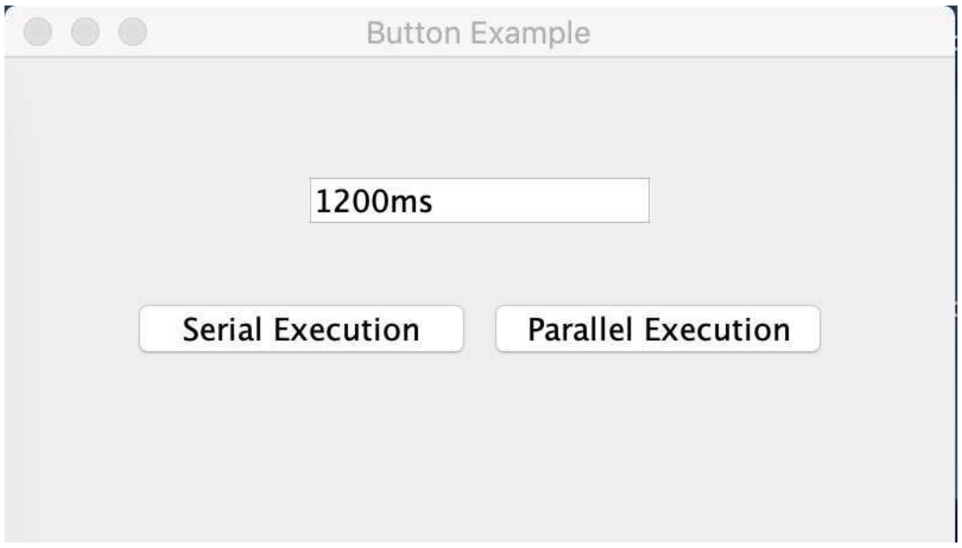


Figure 6: Serial execution time shown in the Applet

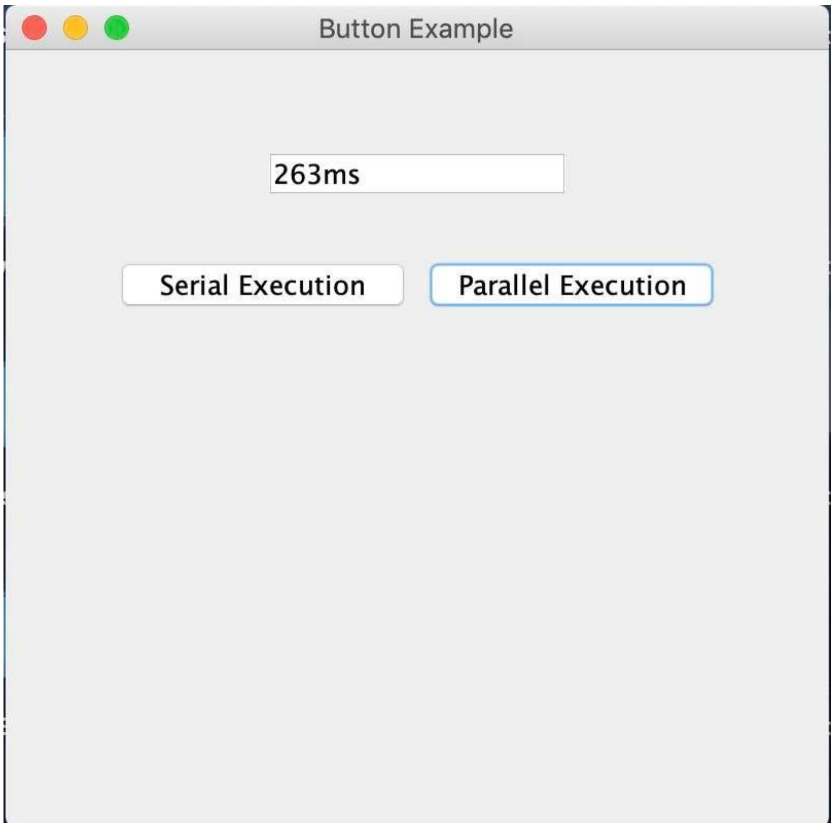
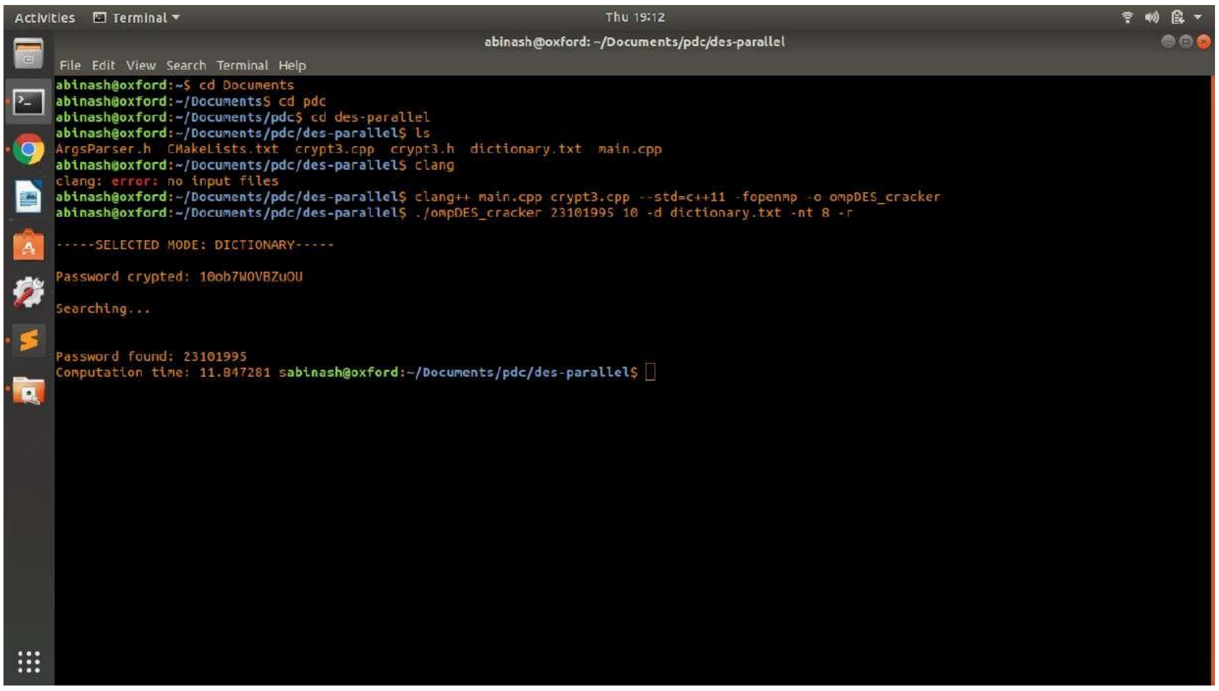


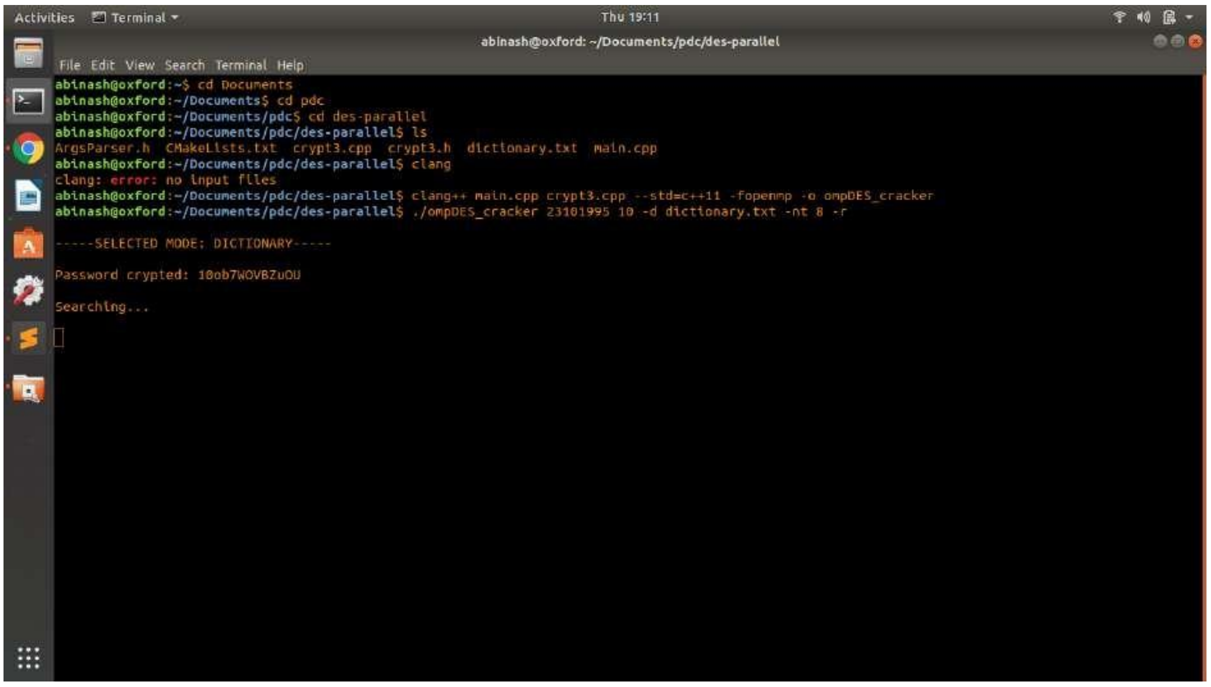
Figure 7: Parallel execution time shown in the Applet

Open MP Implementation


Activities
Terminal
Thu 19:12
abinash@oxford: ~/Documents/pdc/des-parallel
File Edit View Search Terminal Help

```
abinash@oxford:~$ cd Documents
abinash@oxford:~/Documents$ cd pdc
abinash@oxford:~/Documents/pdc$ cd des-parallel
abinash@oxford:~/Documents/pdc/des-parallel$ ls
ArgsParser.h  CMakeLists.txt  crypt3.cpp  crypt3.h  dictionary.txt  main.cpp
abinash@oxford:~/Documents/pdc/des-parallel$ clang
clang: error: no input files
abinash@oxford:~/Documents/pdc/des-parallel$ clang++ main.cpp crypt3.cpp --std=c++11 -fopenmp -o ompDES_cracker
abinash@oxford:~/Documents/pdc/des-parallel$ ./ompDES_cracker 23101995 10 -d dictionary.txt -nt 8 -r

-----SELECTED MODE: DICTIONARY-----

Password crypted: 10ob7WOVBZuOU

Searching...


Password found: 23101995
Computation time: 11.847281 sabinash@oxford:~/Documents/pdc/des-parallel$ 
```

Figure 8 : OpenMP Decryption


Activities
Terminal
Thu 19:11
abinash@oxford: ~/Documents/pdc/des-parallel
File Edit View Search Terminal Help

```
abinash@oxford:~$ cd Documents
abinash@oxford:~/Documents$ cd pdc
abinash@oxford:~/Documents/pdc$ cd des-parallel
abinash@oxford:~/Documents/pdc/des-parallel$ ls
ArgsParser.h  CMakeLists.txt  crypt3.cpp  crypt3.h  dictionary.txt  main.cpp
abinash@oxford:~/Documents/pdc/des-parallel$ clang
clang: error: no input files
abinash@oxford:~/Documents/pdc/des-parallel$ clang++ main.cpp crypt3.cpp --std=c++11 -fopenmp -o ompDES_cracker
abinash@oxford:~/Documents/pdc/des-parallel$ ./ompDES_cracker 23101995 10 -d dictionary.txt -nt 8 -r

-----SELECTED MODE: DICTIONARY-----

Password crypted: 10ob7WOVBZuOU

Searching...

```

Figure 9: Open MP encryption

7. CONCLUSION

Clearly parallel execution time is less than serial execution time by a margin of 937ms in RMI parallelization which is a significant difference in time. But the difference in serial and OpenMP parallelization is 1188.153ms which shows OpenMP parallelization is more efficient. OpenMP parallelization is more efficient than RMI by a margin of 251.153 ms

DES Algorithm is computationally expensive and when implemented in a single processor environment takes a few seconds to encrypt even a small phrase like 'This is a test phrase.' When implemented in a client and server based scenario using Remote Method Invocation (RMI), time taken for encryption is nearly ten times faster. This not only shows the computational power that RMI gives to a programmer but also justifies the motive of the study of parallelizing the code for faster results.

8. REFERENCES

[1] Vishal Pachori; Gunjan Ansari; Neha Chaudhary. Improved Performance of Advance Encryption Standard using Parallel Computing. International Journal of Engineering Research and Applications (IJERA),
Vol. 2, Issue 1,Jan-Feb 2012, pp.967-971

[2] V. Beletskyy, D. Burak. Parallelization of the Data Encryption Standard (DES) algorithm. Website:
http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.511.4739&rep=rep1&type=pdf

[3] Pallavi S. Shendekar; Vijay S. Gulhane. Task Parallelism using Distributed Computing for Encryption and Decryption. International Journal of Scientific and Research Publications, Volume 4, Issue 6, June 2014

[4] Gurpreet Singh; Supriya. A Study of Encryption Algorithms (RSA, DES, 3DES and AES) for Information Security. International Journal of Computer Applications (0975 – 8887) Volume 67– No.19, April 2013

[5] Nirmaljeet Kaur; Sukhman Sodhi. Data Encryption Standard Algorithm (DES) for Secure Data Transmission. International Journal of Computer Applications (0975 – 8887)

[6] CRS Bhardwaj. Modification Of Des Algorithm. International Journal of Innovative Research and Development. Vol 1 Issue 9, 495-505

[7] Syed Naveel Habib; Rizwan Awan; Waleej Haider. A Modified Simplified Data Encryption Standard Algorithm. International Journal of Computer Science and Software Engineering (IJCSSE), Volume 6, Issue 7,
July 2017, 152-154

[8] Ratnadewi et al 2018 J. Phys.: Conf. Ser. 954 012009

[9] Yang, Yoon Seok & Ho Bahn, Jun & Eun Lee, Seung & Bagherzadeh, Nader. (2009). Parallel and Pipeline Processing for Block Cipher Algorithms on a Network-on-Chip. 849 - 854. 10.1109/ITNG.2009.163.

[10] Mohammed A. Hameed; Ahmed I. Jaber; dr Jamhoor M.Alobaidy; Alaa A. Hajer. Design and Simulation DES Algorithm of Encryption for Information Security. American Journal of Engineering Research (AJER),
Volume-7, Issue-4, pp-13-22

[11] http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.680.393&rep=rep1&type=pdf

[12] Akanksha Mathur. A Research paper: An ASCII value based data encryption algorithm and its comparison with other symmetric data encryption algorithms. International Journal on Computer Science and Engineering (IJCSE), Vol. 4 No. 09 Sep 2012, 1650-1657

[13] Alt M., Gorlatch S. (2003) Future-Based RMI: Optimizing Compositions of Remote Method Calls on the Grid. In: Kosch H., Böszörményi L., Hellwagner H. (eds) Euro-Par 2003 Parallel Processing. Euro-Par 2003.
Lecture Notes in Computer Science, vol 2790. Springer, Berlin, Heidelberg

[14] Poo D., Kiong D., Ashok S. (2008) Object Serialization and Remote Method Invocation. In: Object-
Oriented Programming and Java. Springer, London

[15] Hunt J., Loftus C. (2003) Java and Remote Method Invocation. In: Guide to J2EE: Enterprise Java. Springer Professional Computing. Springer, London