CrackIt - A Distributed Password Cracking System

CSE4001 – Parallel and Distributed Computing

PROJECT BASED COMPONENT REPORT

by

Bella Babu (20BCE0558)

Aadharsh S (20BCE0562)

Anna Jai Joseph (20BCE2597)

Samruddhi Bhalerao (20BCE2224)

Ruksana Tabassum(20BCE2650)

School of Computer Science and Engineering

VIT

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

APRIL 2023

DECLARATION

I hereby declare that the report entitled "**CrackIt - A Distributed Password Cracking System"** submitted by me, for the CSE4001 Parallel and Distributed Computing (EPJ) to Vellore Institute of Technology is a record of bonafide work carried out by me under the supervision of **Dr. N. Narayanan Prasanth.**

I further declare that the work reported in this report has not been submitted and will not be submitted, either in part or in full, for any other courses in this institute or any other institute or university.

Place : Vellore

Date : 5/4/2023

Signature of the Candidate

Bella Babu

Aadharsh S

Anna Jai Joseph

Samruddhi Bhalerao

Ruksana Tabassum

ACKNOWLEDGEMENT

We would like to express our profound gratitude to Mr. Ramesh Babu, Dean of SCOPE department for allowing us to complete our project. We would like to express our special thanks to our mentor Mr. , of SCOPE department for his time and efforts he provided throughout the semester. Your useful advice and suggestions were really helpful to us during the project’s completion. In this aspect, we are eternally grateful to you. We would also like to extend our thanks to our family and friends for their support, guidance and motivation without which we couldn't have completed the project. We would like to acknowledge that this project was completed entirely by our group

ABSTRACT

In cryptanalysis and computer security, nowadays the data is secured using a variety of Firewalls/Locks, Passwords and Encryption which doesn't allow external users to access the data. But sometimes higher security also needs breaching ethically by which we can check the run-time analysis and analyze the Data.

Real Time Examples like the defense sector where there is a procedure of ethical hacking and password breaking to keep in touch with the status of extremis. Password cracking is the process which of recovering passwords from data that have been stored in or transmitted by a computer system

In this Project, we implement a distributed password-cracking system. In order to decipher a password, we will use brute force (testing for all possible combinations) and as the number of combinations increases exponentially with the size of the password, we will need a distributed system to help us in this task. One way to encrypt a password is by using a cryptographic hash function. As the name suggests, a hash function takes an input and returns a fixed-size alphanumeric string. Ideally, it should be highly computationally difficult to regenerate the password given only the hashed text. For our project, we would use the Ubuntu Linux distribution which uses the SHA-512 algorithm to store the encrypted passwords.

CONTENTS

List of Figures

List of Tables

1. INTRODUCTION

Passwords are a commonly used method of authentication that help to verify the identity of a user. They are used to protect sensitive information and prevent unauthorized access to accounts, devices, and systems. Passwords are necessary for the following reasons:

- Security: Passwords provide a layer of security that helps to keep unauthorized users out of accounts or systems. Without passwords, anyone could access your personal information or take control of your accounts.
- Privacy: Passwords help to protect your privacy by keeping your information confidential. They prevent others from accessing your personal data, such as financial information, personal correspondence, or other sensitive data.
- Accountability: Passwords are also used to create accountability, as they provide a way to track who has access to an account or system. This can be important for auditing and compliance purposes.
- Personalization: Passwords allow users to personalize their accounts and create a unique identity online. This can be important for maintaining a sense of ownership and control over online activities.

Distributed computing refers to a computing paradigm in which a task is divided into smaller sub-tasks and distributed across multiple computers or nodes. Each node works on its assigned sub-task independently and communicates with other nodes to coordinate and exchange information. The goal of distributed computing is to improve performance, scalability, and fault tolerance.

A password cracking system is a tool or software program designed to attempt to guess or crack passwords used to protect accounts, devices, or systems. These systems use a variety of techniques to crack passwords, including brute-force attacks, dictionary attacks, and hybrid attacks.

Brute-force attacks involve trying every possible combination of characters until the correct password is found. This can be a time-consuming process, especially for complex passwords. Dictionary attacks use a list of commonly used passwords or words found in a dictionary to try and guess the password. This is often faster than brute-force attacks, as it limits the number of possible combinations to try. Hybrid attacks combine elements of both brute-force and dictionary attacks. They use variations of dictionary words, such as adding numbers or symbols to them, to increase the number of possible password combinations.

Password cracking systems are often used by hackers and cybercriminals to gain unauthorized access to accounts, devices, or systems. They can also be used by security professionals to test the strength of passwords and identify weaknesses in security systems. However, it is important to note that using password cracking systems to gain unauthorized access is illegal and can result in severe legal consequences. There are some ethical uses of password cracking systems that are aimed at improving security and protecting against unauthorized access.

- Penetration Testing: Security professionals use password cracking systems during penetration testing to test the security of a system or network. They attempt to crack passwords to identify vulnerabilities and weak points in the security infrastructure, and then recommend measures to address them.
- Password Recovery: Sometimes users forget their passwords or lose access to their accounts, and password cracking systems can be used by authorized individuals or IT departments to recover the lost passwords. This can be particularly useful in situations where data is at risk of being lost due to inaccessible accounts.
- Digital Forensics: Password cracking systems are often used during digital forensics investigations to extract data from encrypted files or to gain access to an account or device used in a crime. This can be used to collect evidence that can help in a criminal investigation.

In this Project, we implement a distributed password-cracking system. In order to decipher a password, we will use brute force (testing for all possible combinations) and as the number of combinations increases exponentially with the size of the password, we will need a distributed system to help us in this task.

One way to encrypt a password is by using a cryptographic hash function. As the name suggests, a hash function takes an input and returns a fixed-size alphanumeric string. Ideally, it should be highly computationally difficult to regenerate the password given only the hashed text. For our project, we would use the Ubuntu Linux distribution which uses the SHA-512 algorithm to store the encrypted passwords. In this project the username and the hash of the password are stored in a file along with the salt string that is used to compute the hash.

The system then uses brute force algorithm to predict the password for each user and this predicted string is hashed using the salt string and this hash is compared with the value stored in the file for a particular user. If there is a match then we can say the password has been cracked.

1.1OBJECTIVES

The objective of a Distributed Password Cracking System is to efficiently and effectively crack password hashes by distributing the workload across multiple machines or nodes in a network. The system can be designed to use various cracking techniques, such as brute-force, dictionary attack, hybrid attack, and rainbow table attack, to crack the passwords. The system can also be optimized to use GPUs or other specialized hardware for faster cracking. The primary goal of the system is to improve the speed and accuracy of password cracking, which can be useful for various security-related purposes. For instance, it can be used by security professionals to test the strength of their organization's passwords or by law enforcement agencies to crack passwords in criminal investigations. However, it's important to note that the use of a password cracking system can also be potentially dangerous and illegal if used for malicious purposes. Therefore, it's crucial to ensure that the system is used ethically and with proper authorization.

1.2MOTIVATION

In cryptanalysis and computer security, Nowadays the Data is Secured using variety of Firewalls/Locks, Passwords and Encryption which doesn't allow external users to access the data. But sometimes higher security also needs breaching ethically by which we can check the run-time analysis and analyze the Data. Real Time Examples like the defense sector where there is a procedure of ethical hacking and password breaking to keep in touch with the status of extremis. Password cracking is the process of recovering passwords from data that have been stored in or transmitted by a computer system. Thus building an effective and high performance password cracking system is essential for security and distributed system helps in increasing the performance of the system by applying brute force parallelly on multiple processors thus reducing computation time.

1.3PROBLEM STATEMENT

In cryptanalysis and computer security, Nowadays the Data is Secured using variety of Firewalls/Locks, Passwords and Encryption which doesn't allow external users to access the data. But sometimes higher security also needs breaching ethically by which we can check the run-time analysis and analyze the Data. Real Time Examples like the defense sector where there is a procedure of ethical hacking and password breaking to keep in touch with the status of extremis.

Password cracking is the process of recovering passwords from data that have been stored in or transmitted by a computer system. Thus building an effective and high performance password cracking system is essential for security and distributed system helps in increasing the performance of the system by applying brute force parallelly on multiple processors thus reducing computation time.

2. LITERATURE REVIEW:

S.No	Title	Findings	Drawbacks
1.	Password cracking with BOINC and hashcat	In the paper, we show how to use BOINC framework to control a network of hashcat- equipped nodes and provide a working solution for performing different cracking attacks. We also provide experimental results of multiple cracking tasks to demonstrate the applicability of our approach. Last but not least, we compare our solution to an existing hashcat-based distributed tool - Hashtopolis.	Hashtopolis is by design more low-level and closer to hashcat, allowing the user to craft attack commands directly, Fitcrack provides higher level of abstraction and automation.
2.	Brute-force and dictionary attack on hashed real- world passwords	Performed a broad targeted attack combining several well-established cracking techniques, such as brute- force, dictionary, and hybrid attacks, on the passwords	GPU can crack over 95% of passwords in just few days, while a more dedicated system can crack all but the strongest 0.5% of them.
3.	Artificial Intelligence-Based Password Brute Force Attacks	This method proposes to use an open-source machine learning algorithm called Torch-rnn, which is available from GitHub, to generate new potential passwords following a similar pattern based on prior passwords and insert them	Defensive strategies to protect our passwords against this new-generation and smarter AI-based password brute force attacks are only coming up

		into the brute force dictionary in real time.	
4.	Password Cracking Using Probabilistic Context-Free Grammars	We first automatically create a probabilistic context-free grammar based upon a training set of previously disclosed passwords. This grammar then allows us to generate word-mangling rules, and from them, password guesses to be used in password cracking.	approach less efficient than John the Ripper password cracker
5.	Distributed Password Cracking using BOINC and John the Ripper	BOINC is a distributed data processing system that incorporates client-server relationships to generically process data. The BOINC structure supports any system that requires large amounts of data to be processed without changing significant portions of the structure. John the Ripper is a password cracking program that takes a password file and attempts to determine the password by a guess and check method. The merger of these two programs enables companies and diverse groups to verify the strength of their password security policy.	The main disadvantage is that John The Ripper password recovery tool is little bit complicated

6.	To Improve Reliability of Message Passing In MPI Libraries Using Flow Checker	The MPI issues a Flow checker to both the sender and the receiver after the verification of the secret key. The generation of the Flow checker involves the selection of 8-bit random key using the appropriate function available in .Net. By using RSA algorithm session key is generated. The session key is converted into binary from which the last two binary digits are chosen through which the Flow checker is created.	Web Browser or other client Program provides credentials in the form of username and Password. Although the scheme is easily implemented, it relies on the assumption that the connection between the client and server computers is secure and can be trusted. The credentials are passed as plaintext and could be intercepted easily. The scheme also provides no protection for the information passed back from the server.
7.	Adversarial Password Cracking	This paper explores password cracking with PassGAN. PassGAN replaces rulebased password guessing, and also password guessing based on data-driven Markov decision systems, with a adversarial methods using deep learning. This is done by training a neural network to determine password attributes autonomously, and using the knowledge of user password attributes to learn and mimic the distribution of previous	Highly depends on the relevancy of the datasets it’s trained on than anything.

		passwords. The advantage that deep neural networks hold is that they can be trained without any a prior knowledge of any properties and structures of user password choices. This makes deep networks more efficient as compared to Markov models which implicitly assume that all relevant password characteristics can be defined in terms of n− grams, and rule-based approaches which can guess only passwords that match with the available rules. As a result, samples generated using a neural network are not limited to a particular subset of the password space. Instead, neural networks can autonomously encode a wide range of password guessing knowledge that includes and surpasses what is captured in human-generated rules and Markovian password generation processes.	
8.	Experiences on Teaching Parallel and Distributed Computing for Undergraduates	This research focuses on launching exhaustive search attacks on authentication mechanisms that use password hashing schemes based on a cryptographic hash function. We focus on how these password schemes can be efficiently implemented on GPU's, which can initiate massive parallel execution paths at low cost, compared to	Since MD5-crypt is memory intensive, memory calls can not be ignored and therefore our implementation will never reach the same level of performance as the MAX or theoretic model.

		a typical CPU. In particular, the password hashing scheme MD5-crypt is reviewed.	
9.	Using Inexpensive Microclusters and Accessible Materials for Cost-Effective Parallel and Distributed Computing Education	We present several different microclusters, each built using a different combination of single board computers (SBCs) as its compute nodes, including various ODROID models, Nvidia's Jetson TK1, Adapteva's Parallella, and the Raspberry Pi. We explore different ways that CS educators are using these systems in their teaching, and describe specific courses in which CS educators have used microclusters. Finally, we present an overview of sources of free PDC pedagogical materials that can be used with microclusters	It was unclear if the Parallella was the best option. Programming the co-processor is not significantly easier than programming a GPU, and some students at the end of the course expressed a desire to have learned CUDA instead. While the Parallella system shows much promise, existing software packages and APIs could use more maturity before the Parallella is really ready for integration into an academic course.
10.	An Overview of Study of Password Cracking	Studies brute-force cracking, dictionary cracking and rainbow table cracking. We found some new techniques such as brute-force cracking based on probability method, Markov models and data mining. High performance computing in password cracking is also studied	Need for the integration of various methods will increase the speed and success probability of password cracking

3. TECHNICAL SPECIFICATIONS

HARDWARE SPECIFICATIONS

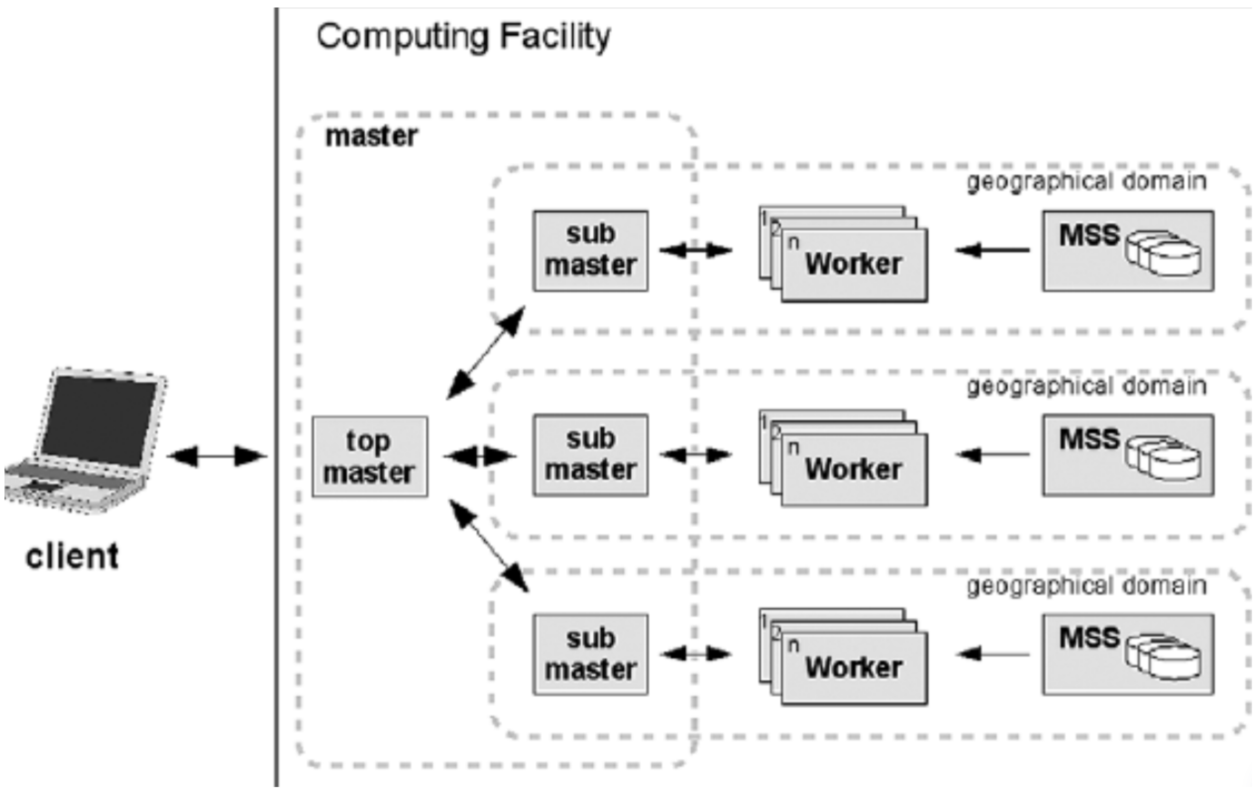
1. Min. 8GB RAM
2. Operating system
3. 15" LCD monitor
4. 500 GB internal storage drive

SOFTWARE SPECIFICATIONS

1. Ubuntu Linux
2. MPI
3. OpenMP
4. C++ Platform
5. Terminal commands

4. DESIGN

Distributed password cracking using OpenMP brute force approach can be implemented by utilizing a master-worker architecture. The architecture involves a master node that manages the distribution of tasks to worker nodes, and worker nodes that perform the password cracking process in parallel.



4.1: Master-Worker Architecture

Here's a high-level architecture of the distributed password cracker using OpenMP brute force approach

1. Master Node: The master node is responsible for managing the distribution of password cracking tasks to worker nodes. It also collects the results from worker nodes and combines them to obtain the final result.

2. Worker Nodes: The worker nodes are responsible for performing the password cracking process. They receive tasks from the master node and execute them in parallel using OpenMP. Once the task is completed, they send the result back to the master node.
3. Password Database: The password database contains the list of hashed passwords that need to be cracked.
4. Brute Force Algorithm: The brute force algorithm is used to generate all possible combinations of passwords that match the given password criteria. OpenMP parallelism is utilized to speed up the process.
5. Result Output: The final result is displayed on the master node, which is a list of cracked passwords.

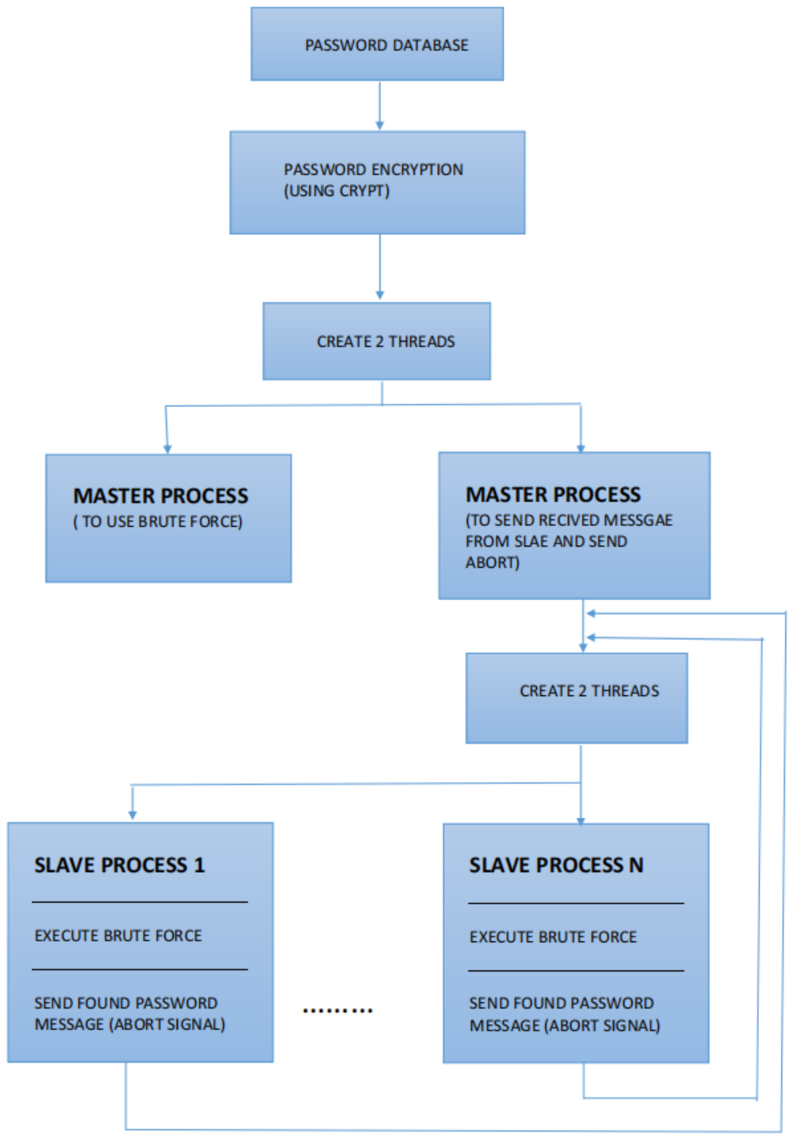
5.PROPOSED SYSTEM

- It is implemented by utilizing a master-worker architecture.
- The encryption of their passwords is done using SHA-512 Algorithm in Ubuntu Linux.
- Password Cracking program is coded in C++ Language by applying Brute Force Algorithm.
- We then parallelize it using MPI and OpenMP.
- Carried out in Ubuntu on Windows.

Steps involved in the password cracking process:

1. The master node reads the password database and generates a list of tasks to be performed by worker nodes.
2. The master node assigns tasks to worker nodes and waits for the results.
3. Worker nodes receive tasks from the master node and perform password cracking in parallel using OpenMP brute force approach.
4. Once the password is cracked, the worker node sends the result back to the master node.
5. The master node collects the results from worker nodes and displays the final list of cracked passwords.

To summarize, the distributed password cracker using OpenMP brute force approach involves a master-worker architecture, where the master node manages the distribution of tasks to worker nodes, and worker nodes perform the password cracking process in parallel using OpenMP. The system uses a brute force algorithm to generate all possible combinations of passwords that match the given password criteria. The final result is displayed on the master node, which is a list of cracked passwords.



5.1: Proposed Architecture for Password cracking

The master and the slaves communicate with each by using MPI parallel programming, while there are 2 threads which each master or slave which are used for cracking the password and for receiving message from other processes. The program uses the crypt() function to produce the hash of the cracked password and check it against the password hash stored in the database to verify that password has been cracked.

6. RESULTS AND DISCUSSIONS

When the program is run, it first asks the user to enter the username for which to crack the password. The code then checks the password database to see if such a user exists and if it does, starts with the parallel brute force approach to crack the password.

Master process cracking password:

```
aadharsh@Aadharsh-S-Alappatt:~$ vi project.cpp
aadharsh@Aadharsh-S-Alappatt:~$ vi pswd.txt
aadharsh@Aadharsh-S-Alappatt:~$ mpic++ -fopenmp project.cpp -lcrypt -o main
aadharsh@Aadharsh-S-Alappatt:~$ mpiexec -n 8 -f machinefile ./main
Enter user name for which to crack password: id
found
Password found! It is -> ab
Master process on Aadharsh-S-Alappatt machine has cracked the password :-)
Aborted all processes on other machines !!!
```

6.1: Master cracks password

Slave process cracking password:

```
aadharsh@Aadharsh-S-Alappatt:~$ vi pswd.txt
aadharsh@Aadharsh-S-Alappatt:~$ mpic++ -fopenmp project.cpp -lcrypt -o main
aadharsh@Aadharsh-S-Alappatt:~$ mpiexec -n 8 -f machinefile ./main
Enter user name for which to crack password: anna
found
Password found! It is -> fm
Slave Aborted all processes on other machines !!!
1 process on Aadharsh-S-Alappatt machine has cracked the password :-)
```

6.2: Slave cracks password

The password database is generated using the following program:

```
#include<stdio.h>
#include<unistd.h>
#include<crypt.h>
#include<fstream>
#include<iostream>
using namespace std;
int main(){
        ofstream file;
        file.open("pswd.txt",ios::app);
        char id[]="fm";
        char salt[]="$6$4GfdWqHx$";
        char *en = crypt(id,salt);
        file<<"anna"<<":"<<en<<":"<<endl;
        return 0;
}
```

6.3: Password Database Generator

The password database once created:

```
bella:$6$4GfdWqHx$iCeygt/iNn8gdv/2uqjznMS30vcCaoL2R/ZQEv.y/1/OTfS5UmoAdlGJtJVHvRckzdNgjxkY96ZqRNIpW36My0:
id:$6$4GfdWqHx$WHg9Y2tyNe2FtY7iGi7w3Ya5FwYDijZdS6c.Y8qyQG05w9WH8nU2PitqVISlExxmWUtaiqhfMRwOOTS8YPr7d/:
heera:$6$4GfdWqHx$ksQDBczFKbf2jyozD2fGwwfh1CXVKrROFtTcVdPE6VV23UimKgc1qg5tLM/jVy4w4rhr5dNMG.AJELFjI2STe.:
```

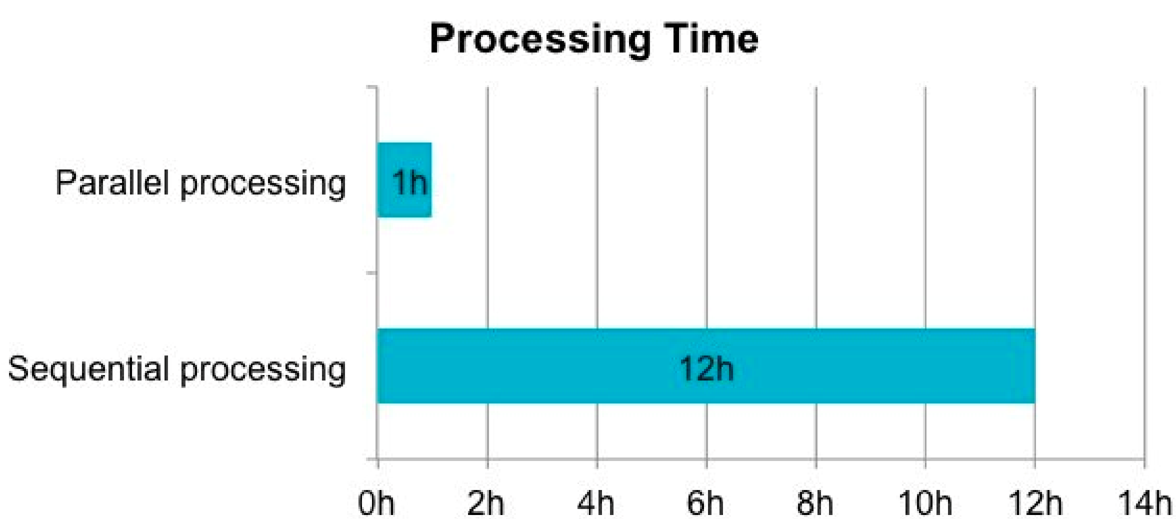
6.4: Hashed Password Database

2: Experimental Results and Analysis

Factor	Sequential Password Cracking	Parallel Brute Force Password Cracking
Speed	Slow, as it checks each password one by one	Fast, as multiple processors check different passwords simultaneously
Cost	Low, as it requires only one processor and little to no additional hardware	High, as it requires multiple processors and possibly additional hardware
Success rate	High, as it can find the correct password eventually	High, as it can try more combinations in less time

Scalability	Limited, as adding more processors doesn't necessarily speed up the process	Highly scalable, as adding more processors can significantly speed up the process
Detection risk	Low, as it generates low traffic and is harder to detect	High, as it generates high traffic and is easier to detect
Efficiency	Inefficient for long and complex passwords	Efficient for long and complex passwords
Resource utilization	Utilizes only one CPU core	Utilizes multiple CPU cores
Time complexity	O(n)	O(n/p) where p is the number of processes.

From the table, we can see that parallel brute force password cracking is generally faster and more efficient for longer and more complex passwords. However, it also comes with a higher cost and detection risk. Sequential password cracking, on the other hand, is slower but has lower costs and detection risk. The choice of which method to use ultimately depends on the specific scenario and resources available.



6.1: The analysis of the result between Sequential and Parallel Processing

7. CONCLUSION:

In conclusion, password cracking through parallel processing using brute force techniques can be a powerful method for breaking passwords. The use of parallel processing allows for multiple passwords to be tested simultaneously, increasing the speed of the process and potentially decreasing the time required to crack a password.

However, the efficiency and speed of password cracking through parallel processing using brute force techniques will depend on several factors such as the number of processors used, the complexity of the password, and the hardware specifications of the machine. A more complex password will require more time to crack than a simpler password, and using more processors may not always guarantee faster cracking times due to potential communication overhead and synchronization issues.

It is important to note that the use of password cracking techniques without proper authorization is illegal and unethical. It is recommended to use these techniques only for legal and ethical purposes, such as in the field of cybersecurity for testing and improving password strength. The strength of passwords can be tested, and possible weaknesses in computer systems can be found, using brute-force password-cracking brute force password cracking techniques. They can, however, also be abused maliciously to obtain unauthorized access to confidential or private data. To avoid unwanted access, both individuals and companies must use is crucial for both individuals and companies to use strong and distinctive passwords, set up two-factor authentication, and frequently update their security policies. The usage of brute force methods should be moral and accountable in order to protect people's privacy and safety, even though they may be helpful for security professionals. Order to protect our online identities and data, we must constantly be on the lookout for threats and take preventative measures.

REFERENCES

1. Radek Hranický, Lukáš Zobal, Ondřej Ryšavý, Dušan Kolář, Distributed password cracking with BOINC and hashcat, Digital Investigation, Volume 30, 2019, Pages 161-172, ISSN 1742-2876.
2. L. Bošnjak, J. Sreš and B. Brumen, "Brute-force and dictionary attack on hashed real-world passwords," 2018 41st International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO), Opatija, Croatia, 2018, pp.1161-1166, doi:10.23919/MIPRO.2018.8400211.
3. Trieu, Khoa and Yang, Yi, "Artificial Intelligence-Based Password Brute Force Attacks" (2018). MWAIS 2018 Proceedings. 39.
4. M. Weir, S. Aggarwal, B. d. Medeiros and B. Glodek, "Password Cracking Using Probabilistic Context-Free Grammars," 2009 30th IEEE Symposium on Security and Privacy, Oakland, CA, USA, 2009, pp. 391-405, doi: 10.1109/SP.2009.8
5. Crumpacker, John R., “Distributed Password Cracking”, 2009 Defense Technical Information Center, NAVAL POSTGRADUATE SCHOOL MONTEREY CA, ADA514270
6. Z. Chen, Q. Gao, W. Zhang and F. Qin, "Improving the Reliability of MPI Libraries via Message Flow Checking," in IEEE Transactions on Parallel and Distributed Systems, vol. 24, no. 3, pp. 535-549, March 2013, doi: 10.1109/TPDS.201 2.127.
7. Nepal, Suman, Isaac Kontomah, Ini Oguntola and Daniel Wang. “Adversarial Password Cracking.” (2019).
8. E. Saule, "Experiences on Teaching Parallel and Distributed Computing for Undergraduates," 2018 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW), Vancouver, BC, Canada, 2018, pp. 361-368, doi: 10.1109/IPDPSW. 2018.00068.
9. F. Yu and Y. Huang, "An Overview of Study of Passowrd Cracking," 2015 International Conference on Computer Science and Mechanical Automation (CSMA), Hangzhou, China, 2015, pp. 25-29, doi: 10.1109/CSMA.20 15.12
10. Adams, Joel & Matthews, Suzanne & Shoop, Elizabeth & Toth, David & Wolfer, James. (2017). Using Inexpensive Microclusters and Accessible Materials for Cost-Effective

Parallel and Distributed Computing Education. The Journal of Computational Science Education. 8. 2-10. 10.22369/issn.2153-4136/8/3/1.

APPENDIX:

pd.cpp:

```
#include<stdio.h>
#include<unistd.h>
#include<crypt.h>
#include<fstream>
#include<iostream>
using namespace std;
int main(){
	ofstream file;
	file.open("pswd.txt",ios::app);
	char id[]="fm";
	char salt[]="$6$4GfdWqHx$";
	char *en = crypt(id,salt);
	file<<"anna"<<":"<<en<<":"<<endl;
	return 0;
}
```

project.cpp:

```
#include <iostream>
#include "mpi.h"
#include <stdlib.h>
#include <string.h>
#include <omp.h>
#include <unistd.h>
#include <stdio.h>
#include <crypt.h>
#include <fstream>
using namespace std;

// function to find all combinations of passwords starting with character passed as 'pswd'
// return empty string if password not found or any other process has found the password
string bruteForce(string pswd, string line, string str_salt, char *&abort_message)
{
	char *encrypted;
	int ind = pswd.length() - 1;

	do
	{
```

```
        if (strcmp(abort_message, "ABORT") == 0) {  // if password is found by any other process
then abort
            return "";
        }

                encrypted = crypt(pswd.c_str(), str_salt.c_str());

                if (!strcmp(encrypted, line.c_str()))       // if password is found then return the password
                {
                        cout << "\nPassword found! It is -> " << pswd << endl;
                        return pswd;
                }
                else
                {
                        if (ind == 0)   // staring with initial character and appending one more character
                        {
                                pswd += 'a';
                                ind = pswd.length() - 1;
                                continue;
                        }

                        pswd[ind] += 1;

                        if (pswd[ind] > 122)    // if character has reached 'z'
                        {
                                int i = ind;
                                while (i > 0)   // back traverse and increment the character
                                {
                    if (strcmp(abort_message, "ABORT") == 0) {  // if password is found by any other
process then abort
                        return "";
                    }

                                        pswd[i] += 1;

                                        if (i > 1 && pswd[i] > 122 && pswd[i-1] != 'z') // brute force - checking
all combinations
                                        {
                                                pswd[i - 1] += 1;
                                                while (i <= ind)
                                                {
                                                        pswd[i] = 'a';
                                                        i++;
                                                }
                                                break;
                                        }
```

```
                    else if (i == 1 && pswd[i] > 122)   // increasing length by appening 'a' at
the end
                    {
                        while (i <= ind)
                        {
                            pswd[i] = 'a';
                            i++;
                        }

                        pswd += 'a';
                        ind = pswd.length() - 1;
                        break;
                    }

                    i--;
                }
            }
        }

    } while (strcmp(encrypted, line.c_str()) && pswd.length() < 9);

    return "";
}

// main function
int main(int argc, char **argv)
{
  int rank, root = 0, nprocs, namelen;
  char processorName[10];

  MPI_Init(&argc, &argv);
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);
  MPI_Comm_size(MPI_COMM_WORLD, &nprocs);
  MPI_Get_processor_name(processorName, &namelen);

  if (nprocs == 1) {
    cout << "Processes must be greater than 1 !!!" << endl;
    return 0;
  }

  int size = 26;
  int alphabets_for_slaves = size / (nprocs - 1);
  int alphabets_for_master = size % (nprocs - 1);

  if (rank == root) {     // master process
```

```
bool check = 0;
string user_name, line, temp, str_salt;

// taking user_name input
cout << "Enter user name for which to crack password: ";
cin >> user_name;

// file reading for /etc/shadow
fstream file("pswd.txt", ios::in);
while (file)
{
   file >> line;
   if (line.length() > user_name.length())
   {
      temp = line.substr(0,user_name.length());
      if (temp == user_name)  // if user-name exists in the file
      {
         check = 1;
         int dollar_count = 0;
         line = line.substr(user_name.length() + 1, line.length());
         cout<<"found";
         int i = 0;
         // extracting salt from the line
         while (i < line.length())
         {
            if (line[i] == '$')
            {
               dollar_count++;
            }

            if (line[i] == ':')
            {
               line = line.substr(0, i);
               break;
            }

            if (dollar_count == 3)
            {
               str_salt = line.substr(0, i+1);
               dollar_count++;
            }
            i++;
         }
         break;
  }
   }
```

```
        }
        //cout<<str_salt;
//      cout<<line;
        if(check==0){
                printf("Username does not exist in file\nhttps://ieeexplore.ieee.org/document/7335069");
                return 0;
            }
        file.close();

        // converting string into char* to send to the slaves
        char *temp_line = new char[line.size() + 1];
             line.copy(temp_line, line.size() + 1);
             temp_line[line.size()] = '\0';

        // converting string into char* to send to the slaves
        char *temp_salt = new char[str_salt.size() + 1];
             str_salt.copy(temp_salt, str_salt.size() + 1);
             temp_salt[str_salt.size()] = '\0';

        // sending sizes and necessary information to slaves
        int starting_index = alphabets_for_master + 1;
        int line_size = line.size() + 1, salt_size = str_salt.size() + 1;
        for (int i = 1, j = 0; i < nprocs; ++i) {
          MPI_Send(&line_size, 4, MPI_INT, i, 1230, MPI_COMM_WORLD);
          MPI_Send(&salt_size, 4, MPI_INT, i, 1231, MPI_COMM_WORLD);
          MPI_Send(temp_line, line_size, MPI_CHAR, i, 1232, MPI_COMM_WORLD);
          MPI_Send(temp_salt, salt_size, MPI_CHAR, i, 1233, MPI_COMM_WORLD);
          MPI_Send(&starting_index, 4, MPI_INT, i, 1234, MPI_COMM_WORLD);
          starting_index += alphabets_for_slaves;
        }

        // making 2 threads - one will call brute force to find password on master if needed - second will receive from slaves if anyone has found the password
        // and then sends abort message to all other processes as well
        char *abort_message = new char[6];
        #pragma omp parallel num_threads(2)
        {
          if (omp_get_thread_num() == 0) {    // to do brute force for master
             for (int i = 0; i < alphabets_for_master; ++i) {
                string alphabet = "";
                alphabet += 97 + i;
                string recv = bruteForce(alphabet, line, str_salt, abort_message);
                if (recv != "") {   // if password has found
                   cout << "Master process on " << processorName << " machine has cracked the password :-)\n";
```

```
                    strcpy(abort_message, "ABORT");
                    break;
                }
            }
        }

        else {  // to receive from slaves if anyone has found the password and then send abort message to all other processes
            MPI_Request recvRequest;
            MPI_Status recvStatus;
            int flag = 0;
            char data[28];

            // receiving from processes
            MPI_Irecv(data, 28, MPI_CHAR, MPI_ANY_SOURCE, MPI_ANY_TAG, MPI_COMM_WORLD, &recvRequest);
            while(!flag) {
                MPI_Test(&recvRequest, &flag, &recvStatus);
                if (strcmp(abort_message, "ABORT") == 0) {
                    strcpy(data, "I have found the number :-)");
                    break;
                }
            }

            // password found so abort other processes
            if (flag) {
                strcpy(abort_message, "ABORT");
            }

            // sending message to all other processes to abort
            if (strcmp(data, "I have found the number :-)") == 0) {
                if (nprocs != 2) {
                    cout << "Aborted all processes on other machines !!!" << endl;
                }
                for (int i = 1; i < nprocs; ++i) {  // telling slaves to abort search
                    MPI_Send(abort_message, 6, MPI_CHAR, i, 1236, MPI_COMM_WORLD);
                }
            }
        }
    }
}

else {    // slave processes

    // receiving sizes from the master
    int starting_index, line_size, salt_size;
```

```
    MPI_Recv(&line_size, 4, MPI_INT, 0, 1230, MPI_COMM_WORLD,
MPI_STATUS_IGNORE);
    MPI_Recv(&salt_size, 4, MPI_INT, 0, 1231, MPI_COMM_WORLD,
MPI_STATUS_IGNORE);

    // initialising arrays and then receiving information i.e. salt, hash from the master
    char *line = new char[line_size], *str_salt = new char[salt_size];
    MPI_Recv(line, line_size, MPI_CHAR, 0, 1232, MPI_COMM_WORLD,
MPI_STATUS_IGNORE);
    MPI_Recv(str_salt, salt_size, MPI_CHAR, 0, 1233, MPI_COMM_WORLD,
MPI_STATUS_IGNORE);
    MPI_Recv(&starting_index, 4, MPI_INT, 0, 1234, MPI_COMM_WORLD,
MPI_STATUS_IGNORE);

    char *abort_message = new char[6];
    #pragma omp parallel num_threads(2) // making two threads, one for cracking passcode and
one for receiving abort message from master
    {
      if (omp_get_thread_num() == 0) {    // thread to crack passcode
        for (int i = 0; i < alphabets_for_slaves; ++i) {    // brute force on all characters
          string alphabet = "";
          alphabet += (96 + i + starting_index);
          string recv = bruteForce(alphabet, line, str_salt, abort_message);
          if (recv != "") {   // if password found then send message to master
            char message[] = "I have found the number :-)";
            MPI_Send(message, 28, MPI_CHAR, 0, 1235, MPI_COMM_WORLD);
            cout << "Slave " << rank << " process on " << processorName << " machine has
cracked the password :-)\n";
            break;
          }

          if (strcmp(abort_message, "ABORT") == 0) {  // if password has found by any other
process then abort                       break;
          }
        }

      } else {    // thread to receive abort message
        MPI_Recv(abort_message, 6, MPI_CHAR, 0, 1236, MPI_COMM_WORLD,
MPI_STATUS_IGNORE);
      }
    }
  }
  MPI_Finalize();
    return 0;}
```