



**Faculty of Computers  
& Artificial Intelligence**



**Benha University**

# **High-Speed Encryption Algorithm Based on Polynomial Roots**

A Senior Project Submitted In Partial Fulfillment Of The Requirements  
For The Degree Of Bachelor Of Computers And Artificial Intelligence.

**Scientific Computing Department**

***Project Team***

1. Mohamed Ahmed Elsayed Emary
2. Abdelfattah Zakaria Abdelfattah Morsy
3. Mohamed Abdelfattah Ahmed Abdelfattah
4. Sara Reda Abdallah Elsheikh
5. Dalia Abdallah Mohamed Mahmoud
6. Shrouk Elsayed Mohamed Abdelmoneim

***Under Supervision of :***

**Dr.El-sayed Badr**

**Benha, June 2024**

## ***Acknowledgement***

---

### **Acknowledgement**

We would like to express our sincere gratitude to our project supervisor, Dr. Sayed Badr, for his invaluable guidance and support throughout the completion of our graduation project. His expertise and insightful feedback were instrumental in shaping our research and refining our approach. We are particularly grateful for his willingness to offer constructive criticism, which challenged us to think critically and improve the quality of our work.

Furthermore, Dr. Badr generously provided us with essential resources that facilitated our research. His encouragement and mentorship played a significant role in motivating us to persevere through challenges and achieve our goals. We are deeply indebted to Dr. Badr for his dedication and support, which have significantly contributed to the success of our graduation project.

## ***Declaration***

---

### **Declaration**

We hereby certify that this material, which we now submit for assessment on the program of study leading to the award of Bachelor of Computers and Artificial Intelligence in (High-Speed Encryption Algorithm Based on Polynomial Roots) is entirely our own work, that we have exercised reasonable care to ensure that the work is original, and does not to the best of our knowledge breach any law of copyright, and has not been taken from the work of others save and to the extent that such work has been cited and acknowledged within the text of our work.

**Signed:** \_\_\_\_\_

**Signed:** \_\_\_\_\_

**Signed:** \_\_\_\_\_

**Signed:** \_\_\_\_\_

**Signed:** \_\_\_\_\_

**Signed:** \_\_\_\_\_

*June 25, 2024*

## **Abstract**

Cryptographic algorithms are pivotal to protecting sensitive data in the digital age, but traditional symmetric encryption methods often face challenges such as computational complexity and slowness. In response, our project presents a novel, pioneering polynomial interpolation-based encryption algorithm designed to boost encryption and decryption speeds. This innovative approach uses polynomials derived from secret keys, employing a secant root-finding method that has proven superior in terms of number of iterations, CPU time, and function value to find the root of the polynomial.

By encrypting messages to ciphertext roots and efficiently decrypting them back to plaintext, our algorithm shows significant performance gains over AES. Furthermore, we have implemented our algorithm in a secure messaging web application called InTouch. The InTouch system ensures end-to-end encryption, where only the sender can encrypt messages before sending them, and only the intended recipient can decrypt and view them using the shared key. This integration not only highlights the efficiency of the algorithm but also emphasizes its practicality in securing communications across different platforms. By leveraging polynomial interpolation and optimizing root-finding operations, our project provides a cryptographic solution that excels in both efficiency and security. This advance holds the potential to facilitate widespread adoption of strong encryption in communications networks

## Contents

<b>1</b>	<b><i>Chapter One: Introduction</i></b>	<b>5</b>
1.1	Introduction . . . . .	5
1.2	Problem & Project Aim . . . . .	5
<b>2</b>	<b><i>Chapter Two: Numerical Methods</i></b>	<b>6</b>
2.1	Root Finding Methods . . . . .	6
2.2	Bisection Method . . . . .	6
2.2.1	How Does the Bisection Method Work? . . . . .	6
2.2.2	Bisection Method Advantages . . . . .	7
2.2.3	Bisection Method Disadvantages . . . . .	7
2.3	False Position Method . . . . .	7
2.3.1	How Does the False Position Method Work? . . . . .	8
2.3.2	False Position Method Advantages . . . . .	8
2.3.3	False Position Method Disadvantages . . . . .	8
2.4	Secant Method . . . . .	8
2.4.1	How Does the Secant Method Work? . . . . .	8
2.4.2	Secant Method Advantages . . . . .	9
2.4.3	Secant Method Disadvantages . . . . .	9
2.5	HybridBF Algorithm . . . . .	10
2.6	HybridSF Algorithm . . . . .	11
2.7	Equations That Serve as Test Cases . . . . .	11
2.8	Extra Equations From Paper . . . . .	12
2.9	Root Finding Algorithms Performance Results . . . . .	12
2.9.1	False Position . . . . .	13
2.9.2	Bisection Method . . . . .	13
2.9.3	Secant Method . . . . .	14
2.9.4	HybridBF Method . . . . .	15
2.9.5	HybridSF Method . . . . .	15
2.9.6	Conclusion . . . . .	16
2.9.6.1	Iterations . . . . .	16
2.9.6.1.1	Bisection, False Position, HybridBF . . . . .	16
2.9.6.1.2	Secant, False Position, HybridSF . . . . .	16
2.9.6.2	CPU Time . . . . .	17
2.9.6.2.1	Bisection, False Position, HybridBF . . . . .	17
2.9.6.2.2	Secant, False Position, HybridSF . . . . .	18
2.9.6.3	Function Value . . . . .	18
2.9.6.3.1	Bisection, False Position, HybridBF . . . . .	18
2.9.6.3.2	Secant, False Position, HybridSF . . . . .	18
2.9.6.4	Secant vs HybridBF . . . . .	19
2.9.6.4.1	Iterations . . . . .	19
2.9.6.4.2	CPU Time . . . . .	19
2.9.6.4.3	Function Value . . . . .	20
<b>3</b>	<b><i>Chapter Three: IBGA Algorithm</i></b>	<b>21</b>
3.1	Introduction . . . . .	21
3.2	IBGA Algorithm Breakdown . . . . .	21
3.2.1	Encryption . . . . .	21
3.2.2	Decryption . . . . .	22
3.3	Algorithm Pseudocode . . . . .	23

3.3.1	Encryption . . . . .	23
3.3.2	Decryption . . . . .	23
3.4	Results . . . . .	24
3.4.1	With Different Root Finding Algorithms . . . . .	24
3.4.1.1	Different Degrees . . . . .	24
3.4.1.1.1	Encryption Time . . . . .	24
3.4.1.1.2	Decryption Time . . . . .	25
3.4.1.1.3	Total Time . . . . .	25
3.4.1.2	Different File sizes . . . . .	26
3.4.1.2.1	Encryption Time . . . . .	26
3.4.1.2.2	Decryption Time . . . . .	26
3.4.1.2.3	Total Time . . . . .	27
3.4.2	Against AES . . . . .	27
3.4.2.1	Different Text Sizes . . . . .	27
3.4.2.1.1	Encryption Time . . . . .	28
3.4.2.1.2	Decryption Time . . . . .	28
3.4.2.1.3	Total Time . . . . .	28
3.4.2.2	Different Polynomial Degrees . . . . .	29
3.4.2.2.1	Encryption Time . . . . .	29
3.4.2.2.2	Decryption Time . . . . .	30
3.4.2.2.3	Total Time . . . . .	30
3.4.3	Final Thoughts . . . . .	30
<b>4</b>	<b>Chapter Four: Design and Analysis</b>	<b>31</b>
4.1	Introduction . . . . .	31
4.2	Functional & Non-Functional Requirements Table . . . . .	31
4.3	Diagrams . . . . .	32
4.3.1	Use Case Diagram . . . . .	32
4.3.2	Activity Diagram . . . . .	33
4.3.3	Class Diagram . . . . .	39
4.3.4	Sequence Diagram . . . . .	40
4.3.5	State Diagram . . . . .	44
4.3.6	Context Diagram . . . . .	45
4.3.7	Data Flow Diagram . . . . .	45
4.3.8	Entity Relationship Diagram . . . . .	46
4.4	Application Screenshots . . . . .	46
4.4.1	Sign Up Page . . . . .	46
4.4.2	Sign In Page . . . . .	47
4.4.3	Home Page . . . . .	47
4.4.4	Team Messages . . . . .	48
4.4.5	Messages in Chat . . . . .	48
4.4.6	Messages Behind the Scenes . . . . .	49
4.4.7	Messages in Stream . . . . .	49
4.5	Tools & Libraries . . . . .	49
4.5.1	 Visual Studio Code . . . . .	49
4.5.2	 Git . . . . .	50
4.5.2.1	 GitHub . . . . .	50
4.5.3	 Python . . . . .	50
4.5.3.1	pandas . . . . .	50
4.5.3.2	matplotlib . . . . .	50

4.5.3.3	numpy . . . . .	50
4.5.3.4	scienceplots . . . . .	50
4.5.3.5	scipy . . . . .	50
4.5.3.6	time . . . . .	51
4.5.3.7	string . . . . .	51
4.5.3.8	random . . . . .	51
4.5.3.9	os . . . . .	51
4.5.3.10	Decimal . . . . .	51
4.5.4	Stream.js . . . . .	51
4.5.5	 HTML5 . . . . .	51
4.5.6	 CSS3 . . . . .	51
4.5.7	 JavaScript . . . . .	51
4.5.8	 Node.js . . . . .	52
4.5.9	 React . . . . .	52
4.5.10	 Markdown . . . . .	52
4.5.11	Mermaid . . . . .	52
4.5.12	Pandoc . . . . .	52
<b>5</b>	<b>References</b>	<b>53</b>

## **List of Figures**

1	HybridBF Steps Flowchart . . . . .	10
2	HybridSF Steps Flowchart . . . . .	11
3	Iterations Comparison Bisection, False Position, HybridBF . . . . .	16
4	Iterations Comparison Secant, False Position, HybridSF . . . . .	17
5	CPU Time Comparison Bisection, False Position, HybridBF . . . . .	17
6	CPU Time Comparison Secant, False Position, HybridSF . . . . .	18
7	Function Value Comparison Bisection, False Position, HybridBF . . . . .	18
8	Function Value Comparison Secant, False Position, HybridSF . . . . .	19
9	Iterations Comparison Secant, HybridBF . . . . .	19
10	CPU Time Comparison Secant, HybridBF . . . . .	20
11	Function Value Comparison Secant, HybridBF . . . . .	20
12	Encryption Steps Flowchart . . . . .	22
13	Decryption Steps Flowchart . . . . .	23
14	Root Finding Algorithms Encryption Time With Different Degrees . . . . .	25
15	Root Finding Algorithms Decryption Time With Different Degrees . . . . .	25
16	Root Finding Algorithms Total Time With Different Degrees . . . . .	26
17	Root Finding Algorithms Encryption Time With Different File Sizes . . . . .	26
18	Root Finding Algorithms Decryption Time With Different File Sizes . . . . .	27
19	Root Finding Algorithms Total Time With Different File Sizes . . . . .	27
20	Encryption Time Comparison . . . . .	28
21	Decryption Time Comparison . . . . .	28
22	Total Time Comparison . . . . .	29
23	Encryption Time Comparison . . . . .	29
24	Decryption Time Comparison . . . . .	30
25	Total Time Comparison . . . . .	30
26	Use Case Diagram . . . . .	32
27	Sign up Activity Diagram . . . . .	33
28	Sign in Activity Diagram . . . . .	34

29	Edit Profile Activity Diagram . . . . .	35
30	Search Message Activity Diagram . . . . .	36
31	Search For Conversation Activity Diagram . . . . .	37
32	Search/Receive Data Activity Diagram . . . . .	38
33	Ban Unban Users Activity Diagram . . . . .	39
34	Class Diagram . . . . .	39
35	Registration Sequence Diagram . . . . .	40
36	Login Sequence Diagram . . . . .	41
37	New Chat Sequence Diagram . . . . .	41
38	Send Message Sequence Diagram . . . . .	42
39	Remove Message Sequence Diagram . . . . .	42
40	Logout Sequence Diagram . . . . .	43
41	State Diagram . . . . .	44
42	Context Diagram . . . . .	45
43	Data Flow Diagram . . . . .	45
44	Entity Relationship Diagram . . . . .	46
45	Sign Up Page . . . . .	46
46	Sign In Page . . . . .	47
47	Home Page . . . . .	47
48	Team Messages . . . . .	48
49	Chat . . . . .	48
50	Messages Behind the Scenes . . . . .	49
51	Messages in Stream . . . . .	49

## **List of Tables**

1	Test Cases Equations . . . . .	11
2	Equations From Paper . . . . .	12
3	False Position . . . . .	13
4	Bisection . . . . .	13
5	Secant . . . . .	14
6	HybridBF . . . . .	15
7	HybridSF . . . . .	15
8	Functional & NonFunctional Requirements . . . . .	31

# **1    *Chapter One: Introduction***

## **1.1    Introduction**

The ever-evolving landscape of cyber threats demands constant innovation in the field of cryptography. Existing encryption algorithms, while providing valuable protection, are often riddled with limitations. Computational complexity can hinder performance, and the rise of quantum computing casts a shadow on the future of established methods. This project presents a groundbreaking departure from tradition, introducing a novel encryption algorithm that leverages the potent combination of polynomials and root finding methods.

This paper delves into the intricate details of the algorithm, meticulously explaining each step of the encryption and decryption processes. We provide a comprehensive analysis of its performance, Comparing it with established methods such as AES, showcasing its significant speed advantage.

## **1.2    Problem & Project Aim**

Encryption, regardless of its application, inevitably requires some processing time. This could be for file encryption on a disk or network encryption via a VPN. The extent of this slowdown is contingent on the encryption algorithms employed and, crucially, the proficiency of the programmer who crafted the encryption and decryption code.

Algorithms with high computational complexity can become performance bottlenecks, particularly for systems necessitating real-time data encryption and decryption. This can adversely affect real-time applications such as video conferencing, secure voice calls, or high-speed data transfers. The processing delays induced by encryption can result in lag, disruptions, and a bad user experience.

The aim of this project is to develop a high-speed encryption algorithm that can be used for real-time applications. The algorithm should be able to encrypt and decrypt data at a much faster rate than traditional encryption algorithms, then it will be used to encrypt and decrypt data in real-time messaging web application.

## 2 Chapter Two: Numerical Methods

### 2.1 Root Finding Methods

At the heart of our innovative encryption algorithm lies a powerful mathematical tool: root finding methods. These methods, while seemingly abstract, play a crucial role in ensuring the security and efficiency of our solution. But before we delve into their specific application, let's unpack what they are and why they hold such significance.

In essence, root finding methods aim to solve the equation  $f(x) = 0$ , where  $f(x)$  is any function. They essentially seek the “roots” of the function, which are the values of  $x$  that make the function evaluate to zero. This seemingly simple task becomes incredibly powerful in cryptography.

In our algorithm, we leverage this power by strategically designing the function  $f(x)$  to incorporate the encryption key as an unknown variable. Through carefully chosen root finding methods, we iteratively approach the function’s roots, and in the decryption process, utilize these roots to recover the original data. The elegance of this approach lies in its inherent security: without knowledge of both the root finding method and how the key is embedded within the function, an attacker would face a near-impossible task of finding the correct roots, keeping your data safe.

However, the importance of root finding methods extends far beyond encryption. They have diverse applications across various fields. In numerical analysis, they are used for solving differential equations and optimization problems among other things. In engineering design, they are crucial for calculating parameters in fields like fluid dynamics and structural analysis. In computer graphics, they are essential for generating realistic images and animations.

Root finding algorithms also have a significant role in machine learning. They are utilized in optimization methods like gradient descent, which is a common technique for training models. The goal of these methods is to minimize a loss function, thereby improving the model’s accuracy.

They also have a vital role in economics and finance. They are used to calculate internal rates of return, solve equilibrium equations in economic models, and find optimal investment strategies.

And finally, we will be using root finding methods to solve the polynomial equations that we will be using in our encryption algorithm.

### 2.2 Bisection Method

The Bisection Method is a straightforward and reliable numerical method used for solving equations in mathematics, particularly in the field of engineering. It solves equations by repeatedly bisecting an interval and then selecting a subinterval in which a root must lie for further processing.

#### 2.2.1 How Does the Bisection Method Work?

If we have a function  $f(x)$  that is continuous on the interval  $[a, b]$  and  $f(a) \cdot f(b) < 0$  (the signals of  $f(x)$  at the ends  $a$  and  $b$  are different), then the function has at least one root in the interval  $[a, b]$ . The Bisection Method works by repeatedly bisecting the interval and then selecting a subinterval in which a root must lie for further processing. The value of  $x$  at the midpoint of the interval is equal to  $\frac{a+b}{2}$ , if  $f(\frac{a+b}{2}) = 0$ , then  $\frac{a+b}{2}$  is the root of the equation. If  $f(a) \cdot f(\frac{a+b}{2}) < 0$ , then the root lies in the interval  $[a, \frac{a+b}{2}]$ , and if  $f(\frac{a+b}{2}) \cdot f(b) < 0$ , then the root lies in the interval  $[\frac{a+b}{2}, b]$ . This process is repeated until we reach the desired accuracy.

The number of iterations required to reach the desired accuracy can be calculated using the formula:

$$n = \lceil \log_2 \left( \frac{b-a}{\epsilon} \right) \rceil \quad (1)$$

Where  $n$  is the number of iterations,  $a$  and  $b$  are the lower and upper bounds of the interval, and  $\epsilon$  is the desired accuracy.

The accuracy of the Bisection Method can be calculated using the formula:

$$\epsilon = \frac{b-a}{2^n} \quad (2)$$

### 2.2.2 Bisection Method Advantages

There are several key advantages to the bisection method:

- Guaranteed convergence. The bracketing approach is known as the bisection method, and it is always convergent.
- Errors can be managed. Increasing the number of iterations in the bisection method always results in a more accurate root.
- Doesn't demand complicated calculations. There are no complicated calculations required when using the bisection method. To use the bisection method, we only need to take the average of two values.
- The bisection method is simple and straightforward to programme on a computer.
- In the case of several roots, the bisection procedure is quick.

### 2.2.3 Bisection Method Disadvantages

There are also some limitations to the bisection method:

- Although the Bisection method's convergence is guaranteed, it is often slow.
- Choosing a guess that is close to the root may necessitate numerous iterations to converge.
- Some equations' roots cannot be found. Because there are no bracketing values, like  $f(x) = x^2$ .
- Its rate of convergence is linear.
- It is incapable of determining complex roots.
- If the guess interval contains discontinuities, it cannot be used.
- It cannot be applied over an interval where the function returns values of the same sign.

## 2.3 False Position Method

In mathematics, the regula falsi, method of false position, or false position method is a very old method for solving an equation with one unknown; this method, in modified form, is still in use. In simple terms, the method is the trial and error technique of using test ("false") values for the variable and then adjusting the test value according to the outcome. This is sometimes also referred to as "guess and check". Versions of the method predate the advent of algebra and the use of equations.

### 2.3.1 How Does the False Position Method Work?

If we have a function  $f(x)$  that is continuous on the interval  $[a, b]$  and  $f(a) \cdot f(b) < 0$  (the signals of  $f(x)$  at the ends  $a$  and  $b$  are different), then the function has at least one root in the interval  $[a, b]$ .

The interval  $[a, b]$  have different signs. The false position method uses two endpoints of the interval  $[a, b]$  with initial values ( $r_0 = a, r_1 = b$ ). The connecting line between the two points  $(r_0, f(r_0))$  and  $(r_1, f(r_1))$  intersects the  $x$ -axis at the next estimate,  $r_2$ . Now, we can determine the successive estimates,  $r_n$  from the following relationship:

$$r_n = r_{n-1} - \frac{f(r_{n-1})(r_{n-1} - r_{n-2})}{f(r_{n-1}) - f(r_{n-2})} \quad (3)$$

### 2.3.2 False Position Method Advantages

There are several key advantages to the false position method:

- Convergence is guaranteed: this method is bracketing method and it is always convergent.
- Error can be controlled: increasing number of iteration always yields more accurate root.
- Does not require derivative: this method does not require derivative calculation.

### 2.3.3 False Position Method Disadvantages

There are also some limitations to the false position method:

- Slow Rate of Convergence: Although convergence of Regula Falsi method is guaranteed, it is generally slow.
- Can not find root of some equations. For example:  $f(x) = x^2$  as there are no bracketing values.
- It has linear rate of convergence.
- It fails to determine complex roots.
- It can not be applied if there are discontinuities in the guess interval.
- It can not be applied over an interval where the function takes values of the same sign.

## 2.4 Secant Method

In numerical analysis, the secant method is a root-finding algorithm that uses a sequence of roots of secant lines to better approximate a root of a function  $f$ . Unlike the method of false position, which keeps one endpoint fixed, the secant method uses two moving points. The secant method can be thought of as a finite-difference approximation of Newton's method. However, the method was developed independently of Newton's method and predates it by many years.

### 2.4.1 How Does the Secant Method Work?

The secant method begins with two initial approximations  $x_0$  and  $x_1$  for the root. These points should ideally be close to the actual root. The method then uses the secant line through the points  $(x_0, f(x_0))$  and  $(x_1, f(x_1))$  to obtain a new approximation  $x_2$ , which is the  $x$ -intercept of this line. The formula for the new approximation is:

$$x_n = x_{n-1} - f(x_{n-1}) \frac{x_{n-1} - x_{n-2}}{f(x_{n-1}) - f(x_{n-2})} \quad (4)$$

This process is repeated until the difference between successive approximations is less than a predetermined tolerance level.

### 2.4.2 Secant Method Advantages

The secant method has several advantages:

- **Efficiency:** It typically converges faster than the method of false position and bisection method.
- **Simplicity:** It does not require the function's derivative, unlike Newton's method.
- **Flexibility:** It can handle a wide range of functions.

### 2.4.3 Secant Method Disadvantages

However, the secant method also has its disadvantages:

- **Convergence is not guaranteed:** If the initial guesses are not close to the actual root, the method may fail to converge.
- **Sensitive to initial guesses:** The closer the initial guesses to the root, the faster the convergence.
- **Possibility of divergence:** If the function is not well-behaved, the secant method can diverge.
- **Multiple roots:** The method may have difficulty distinguishing between multiple roots that are close to each other.
- **Requires two initial values:** Unlike the bisection method, which only needs a single interval, the secant method requires two initial approximations.

## 2.5 HybridBF Algorithm

The HybridBF algorithm is a hybrid algorithm between the bisection method and false position method. The algorithm works as follows:

1. Take the polynomial and the interval that contains the root.
2. In each iteration, the algorithm will apply the bisection method and the false position method and get the root from each method.
3. The algorithm will choose the root that will give the smallest absolute value of the polynomial  $|f(x)|$ .
4. The algorithm will stop when the absolute value of the polynomial is less than a certain tolerance we define.

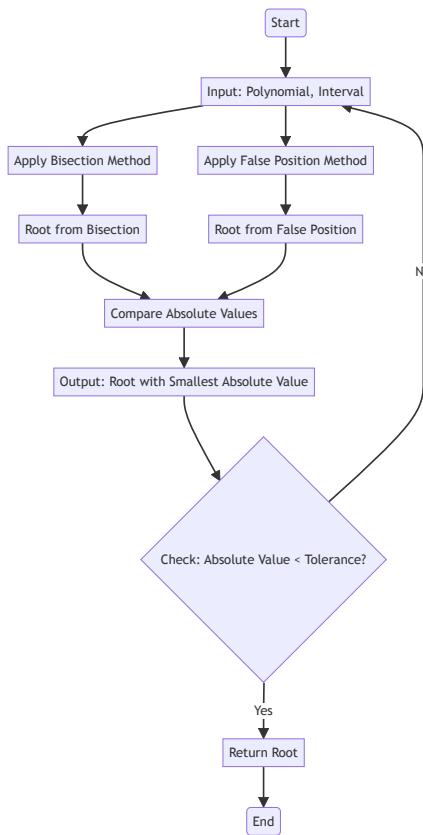


Figure 1: HybridBF Steps Flowchart

## 2.6 HybridSF Algorithm

It has the same steps as HybridBF but it uses the secant method instead of the bisection method.

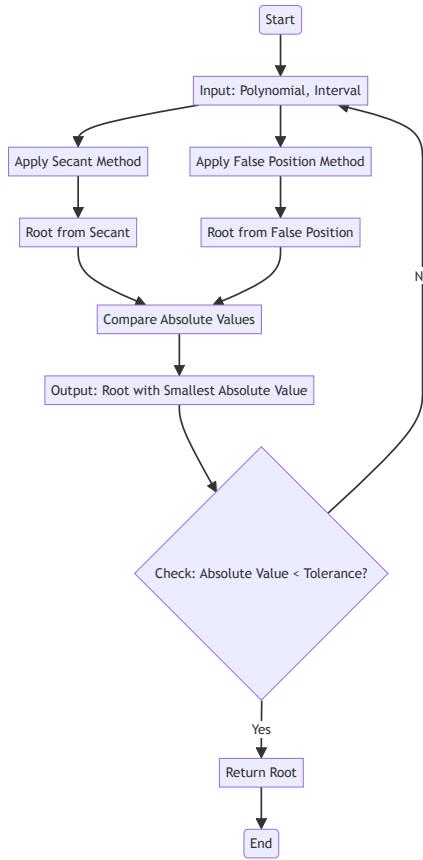


Figure 2: HybridSF Steps Flowchart

To test the algorithm we have used the same 25 equations with each method and run each method (Bisection, False Position, HybridBF, Secant, and HybridSF) 500 times for each problem and then we have calculated the average time and the number of iterations each method have taken for each problem.

We have also used the same tolerance for each method which is  $\epsilon = 10^{-14}$

These are the equations that we have used with each method:

## 2.7 Equations That Serve as Test Cases

In these equations we have tried to use different types of functions like polynomial, exponential, trigonometric, and logarithmic functions to ensure that the algorithm works with different types of functions. We have also used different intervals with each algorithm depending on where the roots of the equations are.

Table 1: Test Cases Equations

No	Equation	Interval
$P_1$	$f(x) = x^3 + 4x^2 - 10 = 0$	$[0, 4]$

No	Equation	Interval
$P_2$	$f(x) = x^2 - 4$	$[0, 4]$
$P_3$	$f(x) = e^x - 2$	$[0, 2]$
$P_4$	$f(x) = \sin(x)$	$[2, 6]$
$P_5$	$f(x) = x^3 - 6x^2 + 11x - 6$	$[1, 2.5]$
$P_6$	$f(x) = x^2 + 3x + 2$	$[-2.5, -1.5]$
$P_7$	$f(x) = \cos(x) - x$	$[0, 1]$
$P_8$	$f(x) = 2^x - 8$	$[2, 4]$
$P_9$	$f(x) = \tan(x)$	$[-1, 1]$
$P_{10}$	$f(x) = x^4 - 8x^3 + 18x^2 - 9x + 1$	$[2, 4]$

## 2.8 Extra Equations From Paper

We got these equations from [this paper](#) and we have used the same intervals too.

Table 2: Equations From Paper

No	Equation	Interval	Reference
$P_{11}$	$f(x) = x^2 - 3$	$[1, 2]$	Harder [18]
$P_{12}$	$f(x) = x^2 - 5$	$[2, 7]$	Srivastava[9]
$P_{13}$	$f(x) = x^2 - 10$	$[3, 4]$	Harder [18]
$P_{14}$	$f(x) = x^2 - x - 2$	$[1, 4]$	Moazzam [10]
$P_{15}$	$f(x) = x^2 + 2x - 7$	$[1, 3]$	Nayak[11]
$P_{16}$	$f(x) = x^3 - 2$	$[0, 2]$	Harder [18]
$P_{17}$	$f(x) = xe^x - 7$	$[0, 2]$	Callhoun [19]
$P_{18}$	$f(x) = x - \cos(x)$	$[0, 1]$	Ehiwario [6]
$P_{19}$	$f(x) = x \sin(x) - 1$	$[0, 2]$	Mathews [20]
$P_{20}$	$f(x) = x \cos(x) + 1$	$[-2, 4]$	Esfandiari [21]
$P_{21}$	$f(x) = x^{10} - 1$	$[0, 1.3]$	Chapra [17]
$P_{22}$	$f(x) = x^2 + e^{x/2} - 5$	$[1, 2]$	Esfandiari [21]
$P_{23}$	$f(x) = \sin(x) \sinh(x) + 1$	$[3, 4]$	Esfandiari [21]
$P_{24}$	$f(x) = e^x - 3x - 2$	$[2, 3]$	Hoffman [22]
$P_{25}$	$f(x) = \sin(x) - x^2$	$[0.5, 1]$	Chapra[17]

## 2.9 Root Finding Algorithms Performance Results

These are the results we got with each method. We have run each method 500 times on each equation and took the average time to get the highest accuracy possible.

### 2.9.1 False Position

These are the results we got with False Position method:

Table 3: False Position

Problem	False Position Algorithm					
	Iter	Avg CPU Time	Approximate Root	Function Value	Lower Bound	Upper Bound
$P_1$	80	0.000108872890472412	1.3652300134141	-7.105427357601E-15	1.3652300134141	4
$P_2$	33	2.54607200622559E-05	2	-8.88178419700125E-15	2	4
$P_3$	51	3.37719917297363E-05	0.693147180559942	-6.21724893790088E-15	0.693147180559942	2
$P_4$	8	4.61483001708984E-06	3.14159265358979	1.22464679914735E-16	3.14159265358979	3.14159265358992
$P_5$	2	2.02751159667969E-06	1	0	1	2.5
$P_6$	31	3.35359573364258E-05	-2	-5.32907051820075E-15	-2.5	-2
$P_7$	12	7.18259811401367E-06	0.739085133215155	9.2148511043888E-15	0.739085133215155	1
$P_8$	30	2.37202644348144E-05	3	-7.105427357601E-15	3	4
$P_9$	2	9.75608825683594E-07	0	0	0	1
$P_{10}$	13	2.66284942626953E-05	3.11174865630925	0	3.11174865630925	3.11174865630925
$P_{11}$	14	1.06868743896484E-05	1.73205080756888	-3.99680288865056E-15	1.73205080756888	2
$P_{12}$	50	3.87668609619141E-05	2.23606797749979	-9.76996261670138E-15	2.23606797749979	7
$P_{13}$	17	1.34563446044922E-05	3.16227766016838	-1.77635683940025E-15	3.16227766016838	4
$P_{14}$	38	3.21516990661621E-05	2	-8.65973959207622E-15	2	4
$P_{15}$	21	2.0256519317627E-05	1.82842712474619	-2.66453525910038E-15	1.82842712474619	3
$P_{16}$	41	3.12848091125488E-05	1.25992104989487	-6.21724893790088E-15	1.25992104989487	2
$P_{17}$	30	2.12483406066895E-05	1.52434520498414	-7.99360577730113E-15	1.52434520498414	2
$P_{18}$	12	7.22360610961914E-06	0.739085133215155	-9.2148511043888E-15	0.739085133215155	1
$P_{19}$	7	5.02967834472656E-06	1.11415714087193	8.88178419700125E-16	1.09975017029462	1.11415714087193
$P_{20}$	13	9.73367691040039E-06	2.07393280909121	7.7715611723761E-16	2.07393280909121	2.51571977101466
$P_{21}$	139	0.000106982231140137	0.9999999999999999	-8.88178419700125E-15	0.9999999999999999	1.3
$P_{22}$	16	1.86800956726074E-05	1.64901326830319	-8.88178419700125E-16	1.64901326830319	2
$P_{23}$	45	5.06892204284668E-05	3.22158839909394	6.43929354282591E-15	3.22158839909394	4
$P_{24}$	45	3.95450592041016E-05	2.12539119881113	-8.88178419700125E-15	2.12539119881113	3
$P_{25}$	17	1.45211219787598E-05	0.876726215395055	7.88258347483861E-15	0.876726215395055	1

### 2.9.2 Bisection Method

These are the results we got with Bisection method:

Table 4: Bisection

Problem	Bisection Algorithm					
	Iter	Avg CPU Time	Approximate Root	Function Value	Lower Bound	Upper Bound
$P_1$	50	3.78618240356445E-05	1.3652300134141	-2.8421709430404E-14	1.36523001341409	1.3652300134141
$P_2$	1	9.20772552490234E-07	2	0	0	4
$P_3$	49	2.00080871582031E-05	0.693147180559944	-3.33066907387547E-15	0.69314718055994	0.693147180559947
$P_4$	50	1.82290077209473E-05	3.14159265358979	1.22464679914735E-16	3.14159265358979	3.1415926535898
$P_5$	48	4.16674613952637E-05	2	0	2	2.000000000000001
$P_6$	1	1.10149383544922E-06	-2	0	-2.5	-1.5
$P_7$	48	1.79662704467773E-05	0.739085133215159	2.55351295663786E-15	0.739085133215156	0.739085133215163
$P_8$	1	9.37938690185547E-07	3	0	2	4
$P_9$	1	7.75814056396484E-07	0	0	-1	1
$P_{10}$	49	5.54814338684082E-05	3.11174865630925	1.06581410364015E-14	3.11174865630925	3.11174865630925
$P_{11}$	48	2.20985412597656E-05	1.73205080756888	4.44089209850063E-15	1.73205080756888	1.73205080756888

Bisection Algorithm						
Problem	Iter	Avg CPU Time	Approximate Root	Function Value	Lower Bound	Upper Bound
$P_{12}$	50	2.28300094604492E-05	2.23606797749979	-1.95399252334028E-14	2.23606797749978	2.23606797749979
$P_{13}$	48	2.22716331481934E-05	3.16227766016838	1.59872115546023E-14	3.16227766016838	3.16227766016839
$P_{14}$	50	2.49624252319336E-05	2	-2.66453525910038E-15	2	2
$P_{15}$	49	2.77295112609863E-05	1.82842712474619	-1.15463194561016E-14	1.82842712474618	1.82842712474619
$P_{16}$	49	2.24394798278809E-05	1.25992104989487	5.32907051820075E-15	1.25992104989487	1.25992104989488
$P_{17}$	49	2.14519500732422E-05	1.52434520498415	3.37507799486048E-14	1.52434520498414	1.52434520498415
$P_{18}$	48	1.79176330566406E-05	0.739085133215159	-2.55351295663786E-15	0.739085133215156	0.739085133215163
$P_{19}$	49	2.24361419677734E-05	1.11415714087193	-2.99760216648792E-15	1.11415714087192	1.11415714087193
$P_{20}$	51	2.31366157531738E-05	2.07393280909122	-1.33226762955019E-15	2.07393280909121	2.07393280909122
$P_{21}$	48	2.20670700073242E-05	1	1.11022302462516E-14	0.9999999999999996	1.000000000000001
$P_{22}$	48	3.11980247497559E-05	1.64901326830319	-3.5527136788005E-15	1.64901326830319	1.64901326830319
$P_{23}$	48	3.00378799438477E-05	3.22158839909394	-5.55111512312578E-15	3.22158839909394	3.22158839909395
$P_{24}$	46	2.39477157592773E-05	2.12539119881113	0	2.12539119881112	2.12539119881114
$P_{25}$	47	2.58064270019531E-05	0.876726215395063	-8.88178419700125E-16	0.87672621539506	0.876726215395067

### 2.9.3 Secant Method

These are the results we got with Secant method:

Table 5: Secant

Secant Algorithm						
Problem	Iter	Avg CPU Time	Approximate Root	Function Value	Lower Bound	Upper Bound
$P_1$	12	6.93511962890625E-06	1.3652300134141	0	1.3652300134141	1.3652300134141
$P_2$	9	3.46088409423828E-06	2	0	2	2
$P_3$	9	3.33023071289062E-06	0.693147180559945	0	0.693147180559945	0.693147180559945
$P_4$	7	2.45857238769531E-06	6.28318530717959	-2.44929359829471E-16	6.28318530717959	6.28318530717959
$P_5$	2	1.84392929077148E-06	1	0	1	1
$P_6$	10	4.70542907714844E-06	-2	0	-2	-2
$P_7$	7	2.44808197021484E-06	0.739085133215161	0	0.739085133215161	0.739085133215161
$P_8$	8	3.28588485717773E-06	3	0	3	3
$P_9$	2	1.04379653930664E-06	0	0	0	0
$P_{10}$	17	1.45139694213867E-05	0.481389601149957	0	0.481389601149957	0.481389601149957
$P_{11}$	7	2.82144546508789E-06	1.73205080756888	4.44089209850063E-16	1.73205080756888	1.73205080756888
$P_{12}$	8	3.25679779052734E-06	2.23606797749979	8.88178419700125E-16	2.23606797749979	2.23606797749979
$P_{13}$	6	2.61688232421875E-06	3.16227766016838	-1.77635683940025E-15	3.16227766016838	3.16227766016838
$P_{14}$	9	3.77416610717773E-06	2	0	2	2
$P_{15}$	7	3.43656539916992E-06	1.82842712474619	8.88178419700125E-16	1.82842712474619	1.82842712474619
$P_{16}$	11	4.30393218994141E-06	1.25992104989487	0	1.25992104989487	1.25992104989487
$P_{17}$	10	3.88193130493164E-06	1.52434520498414	0	1.52434520498414	1.52434520498414
$P_{18}$	7	2.60496139526367E-06	0.739085133215161	0	0.739085133215161	0.739085133215161
$P_{19}$	6	2.66456604003906E-06	1.11415714087193	2.22044604925031E-16	1.11415714087193	1.11415714087193
$P_{20}$	9	3.63922119140625E-06	2.07393280909121	-2.22044604925031E-16	2.07393280909121	2.07393280909121
$P_{21}$	<b>No Solution Was Found</b>					
$P_{22}$	7	4.0740966796875E-06	1.64901326830319	0	1.64901326830319	1.64901326830319
$P_{23}$	8	4.34207916259766E-06	3.22158839909394	3.33066907387547E-16	3.22158839909394	3.22158839909394
$P_{24}$	8	3.73697280883789E-06	2.12539119881113	0	2.12539119881113	2.12539119881113
$P_{25}$	8	3.88669967651367E-06	0.876726215395062	0	0.876726215395062	0.876726215395062

### 2.9.4 HybridBF Method

These are the results we got with HybridBF method:

Table 6: HybridBF

Problem	HybridBF Algorithm					
	Iter	Avg CPU Time	Approximate Root	Function Value	Lower Bound	Upper Bound
$P_1$	10	1.51724815368652E-05	1.3652300134141	-7.105427357601E-15	1.36523001341378	1.36750019802744
$P_2$	1	9.86576080322266E-07	2	0	0	4
$P_3$	10	1.01175308227539E-05	0.693147180559945	0	0.693147180559933	0.695162706464415
$P_4$	6	5.41830062866211E-06	3.14159265358979	1.22464679914735E-16	3.14159035795569	3.14159265360489
$P_5$	1	1.57594680786133E-06	1	0	1	2.5
$P_6$	1	1.17969512939453E-06	-2	0	-2.5	-1.5
$P_7$	8	7.32851028442383E-06	0.739085133215161	1.11022302462516E-16	0.739085133215147	0.742227073217592
$P_8$	1	1.02519989013672E-06	3	0	2	4
$P_9$	1	8.06331634521484E-07	0	0	-1	1
$P_{10}$	8	1.61228179931641E-05	3.11174865630925	0	3.10853799278589	3.11174865630925
$P_{11}$	8	8.37850570678711E-06	1.73205080756888	-4.44089209850063E-16	1.7320508075688	1.73505784022098
$P_{12}$	10	1.055908203125E-05	2.23606797749979	-3.5527136788005E-15	2.23606797749936	2.24392915398361
$P_{13}$	8	8.51917266845703E-06	3.16227766016838	1.77635683940025E-15	3.16227766016837	3.1672187190124
$P_{14}$	2	2.20155715942383E-06	2	0	1.5	2.5
$P_{15}$	5	6.05535507202148E-06	1.82842712474619	0	1.828427124743	1.82842712474938
$P_{16}$	9	9.37175750732422E-06	1.25992104989487	-3.99680288865056E-15	1.25992104989398	1.2611286403177
$P_{17}$	11	1.14202499389648E-05	1.52434520498414	0	1.52434520498414	1.52603333710876
$P_{18}$	8	7.33852386474609E-06	0.739085133215161	-1.11022302462516E-16	0.739085133215147	0.742227073217592
$P_{19}$	6	6.33621215820313E-06	1.11415714087193	-2.22044604925031E-16	1.11324273276427	1.11415714087198
$P_{20}$	10	1.05037689208984E-05	2.07393280909121	-2.22044604925031E-16	2.07393280909119	2.07893500333739
$P_{21}$	12	1.24101638793945E-05	1	-1.11022302462516E-15	0.99999999999993	1.00034336328299
$P_{22}$	8	1.04570388793945E-05	1.64901326830319	-3.5527136788005E-15	1.64901326830264	1.65315575626948
$P_{23}$	9	1.19032859802246E-05	3.22158839909394	3.33066907387547E-16	3.22158839909392	3.22241688813951
$P_{24}$	9	1.03793144226074E-05	2.12539119881113	-6.21724893790088E-15	2.1253911988104	2.12751913344632
$P_{25}$	7	7.85541534423828E-06	0.876726215395058	4.77395900588817E-15	0.876726215388671	0.877268445434873

As we see from the table above the hybrid method tend to be faster and take much less iterations than both Bisection and False Position methods.

### 2.9.5 HybridSF Method

These are the results we got with HybridSF method:

Table 7: HybridSF

Problem	HybridSF Algorithm					
	Iter	Avg CPU Time	Approximate Root	Function Value	Lower Bound	Upper Bound
$P_1$	40	4.32667732238769E-05	1.3652300134141	-7.105427357601E-15	1.3652300134141	4
$P_2$	17	1.1502742767334E-05	2	-2.66453525910038E-15	2	4
$P_3$	26	1.56879425048828E-05	0.693147180559944	-3.10862446895044E-15	0.693147180559942	2
$P_4$	6	3.37457656860352E-06	3.14159265358979	1.22464679914735E-16	3.14159265358979	3.14159265363241
$P_5$	1	1.11484527587891E-06	1	0	1	2.5
$P_6$	16	1.37491226196289E-05	-2	-4.44089209850063E-15	-2.000000000000001	-1.75
$P_7$	6	3.53384017944336E-06	0.739085133215155	9.2148511043888E-15	0.73908513321505	1
$P_8$	15	1.04451179504395E-05	3	-7.105427357601E-15	3	4

Problem	Hybrid Algorithm					
	Iter	Avg CPU Time	Approximate Root	Function Value	Lower Bound	Upper Bound
$P_9$	1	6.39915466308594E-07	0	0	-1	1
$P_{10}$	7	1.10006332397461E-05	3.11174865630925	7.105427357601E-15	3.11174865630925	3.11174865630925
$P_{11}$	7	4.94670867919922E-06	1.73205080756888	-3.99680288865056E-15	1.73205080756886	2
$P_{12}$	25	1.68638229370117E-05	2.23606797749979	-9.76996261670138E-15	2.23606797749979	7
$P_{13}$	9	6.21271133422852E-06	3.16227766016838	1.77635683940025E-15	3.16227766016838	4
$P_{14}$	19	1.37357711791992E-05	2	-8.65973959207622E-15	1.99999999999999	4
$P_{15}$	11	9.09614562988281E-06	1.82842712474619	0	1.82842712474619	3
$P_{16}$	21	1.41358375549316E-05	1.25992104989487	-3.10862446895044E-15	1.25992104989487	2
$P_{17}$	15	9.67836380004883E-06	1.52434520498414	-7.99360577730113E-15	1.52434520498414	2
$P_{18}$	6	3.55148315429688E-06	0.739085133215155	-9.2148511043888E-15	0.73908513321505	1
$P_{19}$	5	3.5557746887207E-06	1.11415714087193	2.22044604925031E-16	1.11415714087193	1.11415714087196
$P_{20}$	7	0.0000048828125	2.07393280909121	1.77635683940025E-15	2.07393280909119	2.51571977101466
$P_{21}$	70	4.62303161621094E-05	1	-6.66133814775094E-15	0.99999999999999	1.3
$P_{22}$	8	7.51972198486328E-06	1.64901326830319	-8.88178419700125E-16	1.64901326830319	2
$P_{23}$	23	2.04296112060547E-05	3.22158839909394	3.33066907387547E-16	3.22158839909394	4
$P_{24}$	23	1.71117782592773E-05	2.12539119881113	-1.77635683940025E-15	2.12539119881113	3
$P_{25}$	9	6.75392150878906E-06	0.876726215395061	1.11022302462516E-15	0.876726215395055	1

## 2.9.6 Conclusion

These are our conclusions based on the tables and the plots:

### 2.9.6.1 Iterations

**2.9.6.1.1 Bisection, False Position, HybridBF** As you see in the plot below the HybridBF method demonstrates superior performance compared to both the bisection and false position methods in terms of the number of iterations required.

As we see here in  $P_{21}$  the false position method have much more number of iterations than both hybrid and bisection methods which will lead to more CPU time as we will see in the next section.

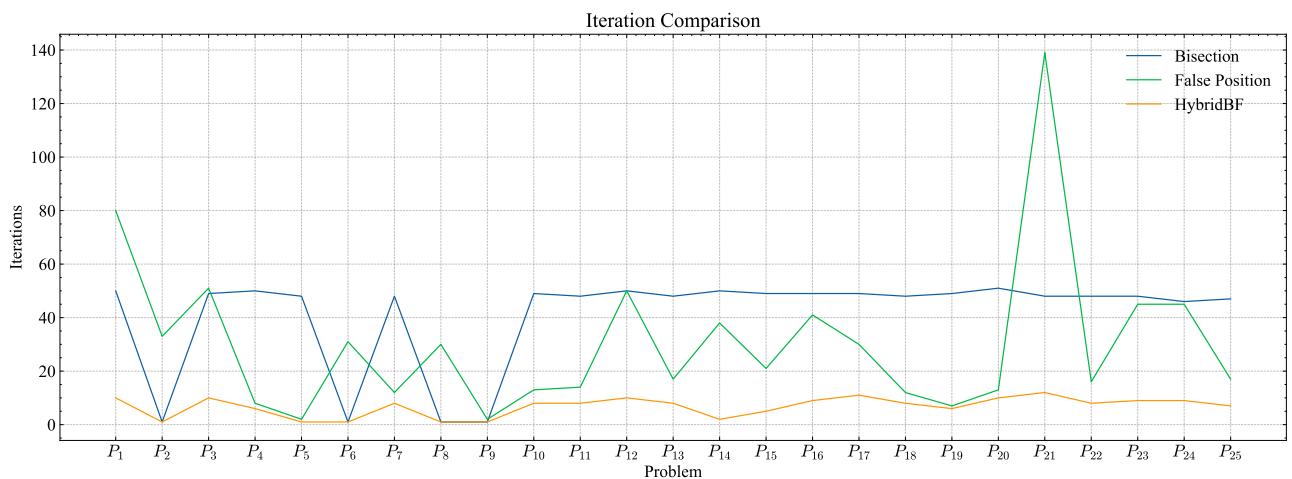


Figure 3: Iterations Comparison Bisection, False Position, HybridBF

**2.9.6.1.2 Secant, False Position, HybridSF** The HybridSF method here didn't have the same performance improvement as in the previous case. The secant method is the fastest in terms of

iterations, followed by the hybrid method then false position, However There are 7 problems where the HybridSF method is faster than the secant method with 1 or 2 iterations, these are  $P_4, P_5, P_7, P_9, P_{18}, P_{19}, P_{20}$ .

You can also notice that the graph of the secant method is **not continuous** on  $P_{21}$  since secant method is not guaranteed to converge, so it wasn't able to find the root in this problem.

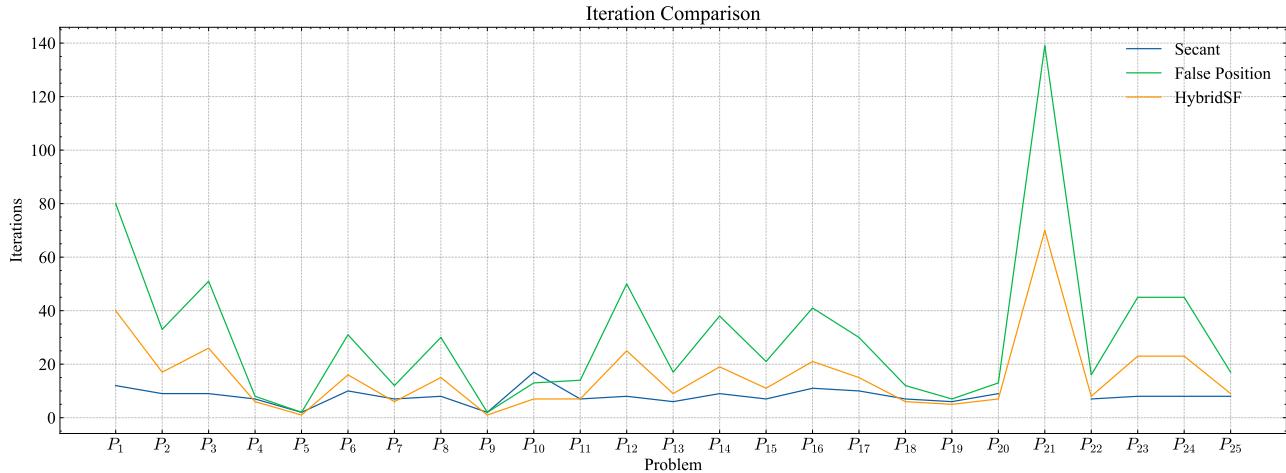


Figure 4: Iterations Comparison Secant, False Position, HybridSF

### 2.9.6.2 CPU Time

**2.9.6.2.1 Bisection, False Position, HybridBF** As a result of having less number of iterations the hybridBF method shows significant improvement over the bisection method in terms of CPU time, with a ratio of 21:4. This translates to approximately 84% for the hybrid method and 16% for the bisection method.

It also shows an improvement over the false position method in terms of CPU time, with a ratio of 19:6. This translates to approximately 76% for the hybrid method and 24% for the false position method.

As a general trend, the hybrid method is faster than both the bisection and false position methods when it comes to finding the approximate root.

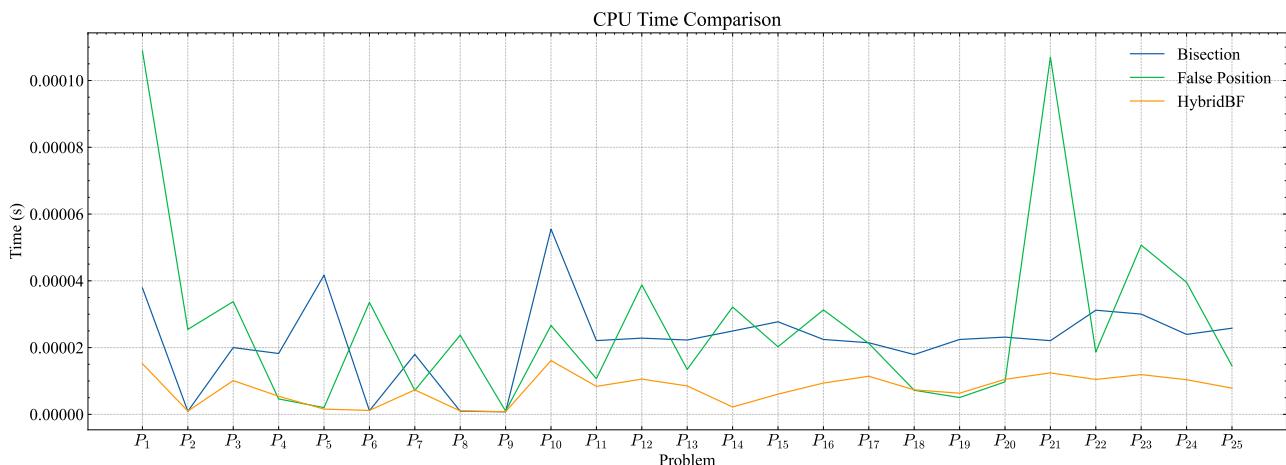


Figure 5: CPU Time Comparison Bisection, False Position, HybridBF

**2.9.6.2.2 Secant, False Position, HybridSF** As a result of secant method having less number of iterations than both HybridSF and false position and much more simple implementation than HybridSF, it has less CPU time. The secant method is the fastest in terms of CPU time, followed by the hybrid method then false position.

This happens in all problems except only two problems which are  $P_5, P_9$  where the HybridSF method is *slightly* faster than the secant method.

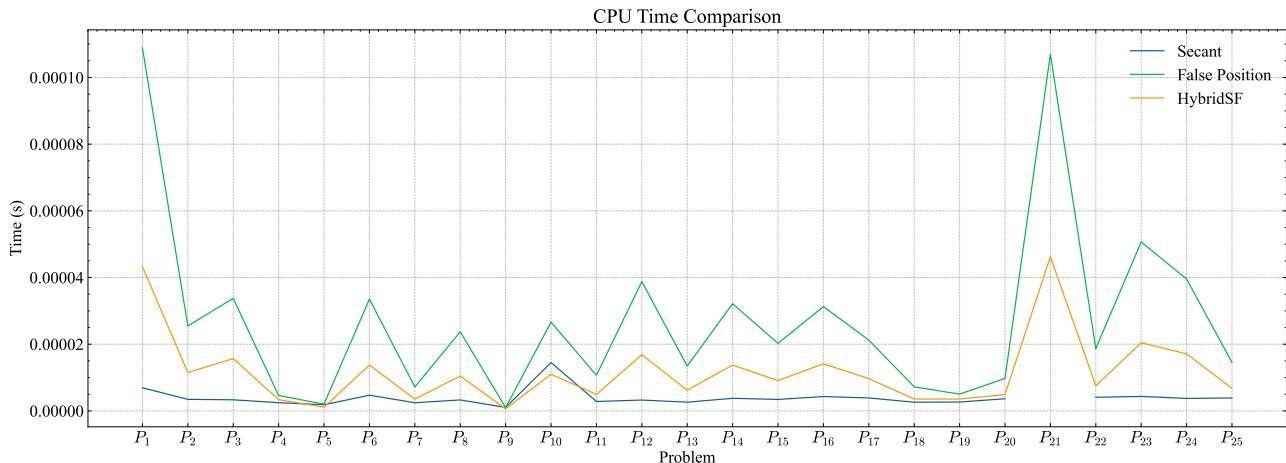


Figure 6: CPU Time Comparison Secant, False Position, HybridSF

### 2.9.6.3 Function Value

**2.9.6.3.1 Bisection, False Position, HybridBF** The hybrid method outperforms both the bisection and false position methods in terms of function value, with smaller absolute values that are closer to zero.

This happens in all problems except only one problem  $P_{24}$  in which the bisection method had the nearest value to zero.

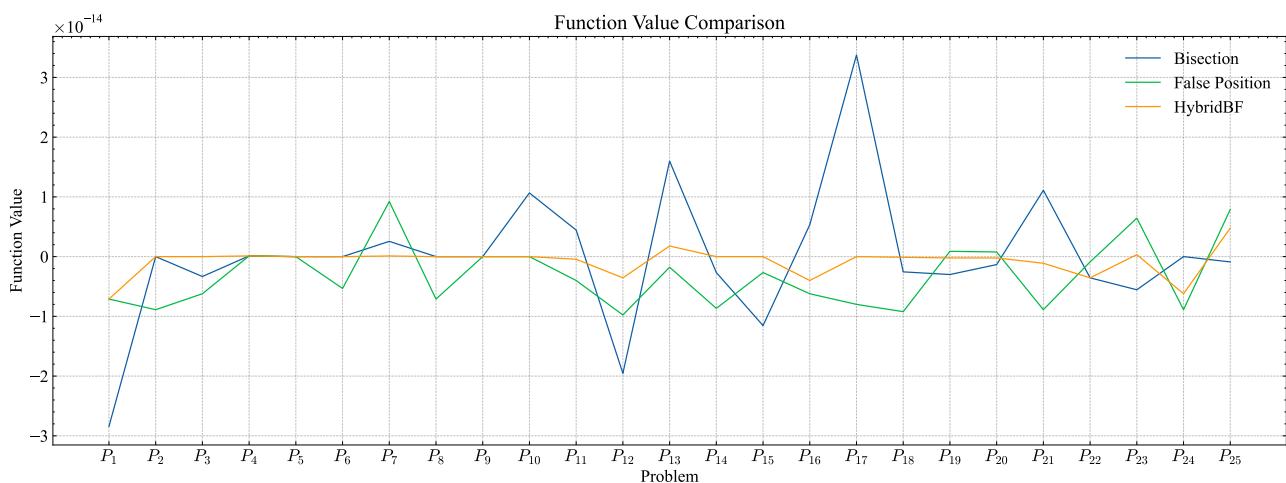


Figure 7: Function Value Comparison Bisection, False Position, HybridBF

**2.9.6.3.2 Secant, False Position, HybridSF** Again the secant method shows an improvement over both the false position and HybridSF methods in terms of function value, with smaller absolute values that are closer to zero.

This happens in all problems except two  $P_4$  in which both HybridSF and false position methods had the nearest value to zero, and  $P_{15}$  in which the HybridSF had a value closer to zero.

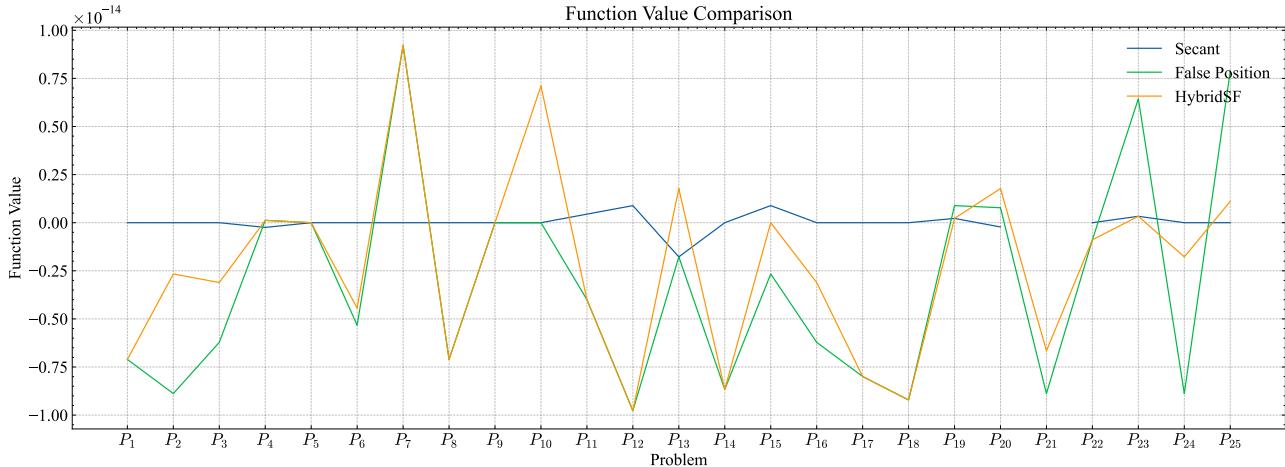


Figure 8: Function Value Comparison Secant, False Position, HybridSF

**2.9.6.4 Secant vs HybridBF** When comparing the results of the fastest two algorithms which are the secant and HybridBF methods, we found the following:

**2.9.6.4.1 Iterations** There is no winner here, the secant method is faster in some problems and the HybridBF method is faster in others but there is a slight advantage for the HybridSF method over the secant method.

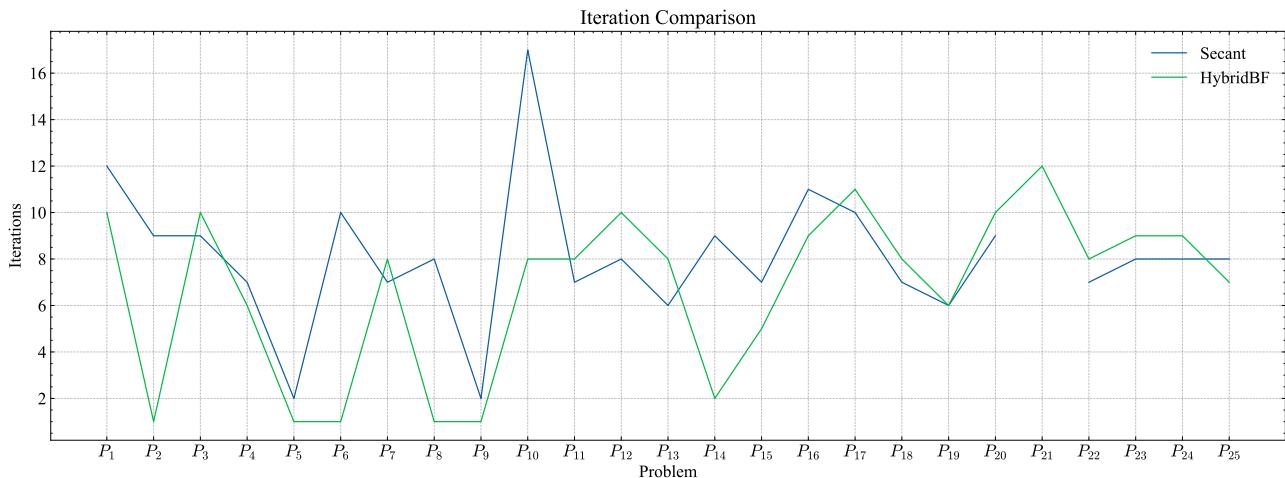


Figure 9: Iterations Comparison Secant, HybridBF

**2.9.6.4.2 CPU Time** When comparing the CPU time of both algorithms we found that the secant method is faster in all problems except for 6 problems which are  $P_2, P_5, P_6, P_8, P_9, P_{14}$ .

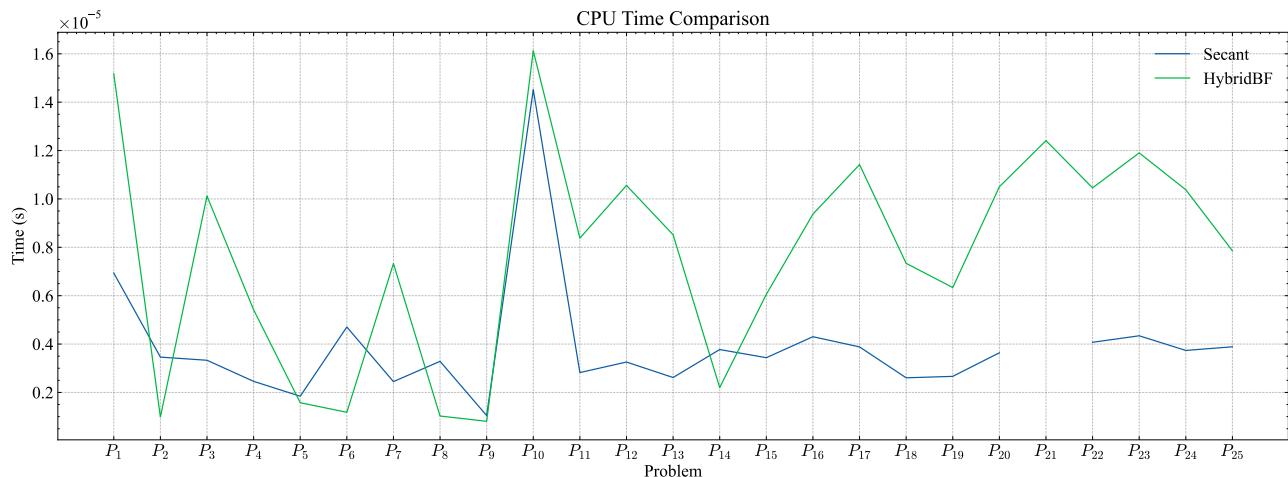


Figure 10: CPU Time Comparison Secant, HybridBF

**2.9.6.4.3 Function Value** When comparing function values with absolute values closer to zero, the results of secant method were better than the HybridBF method in nearly all problems except for  $P_{15}$ .

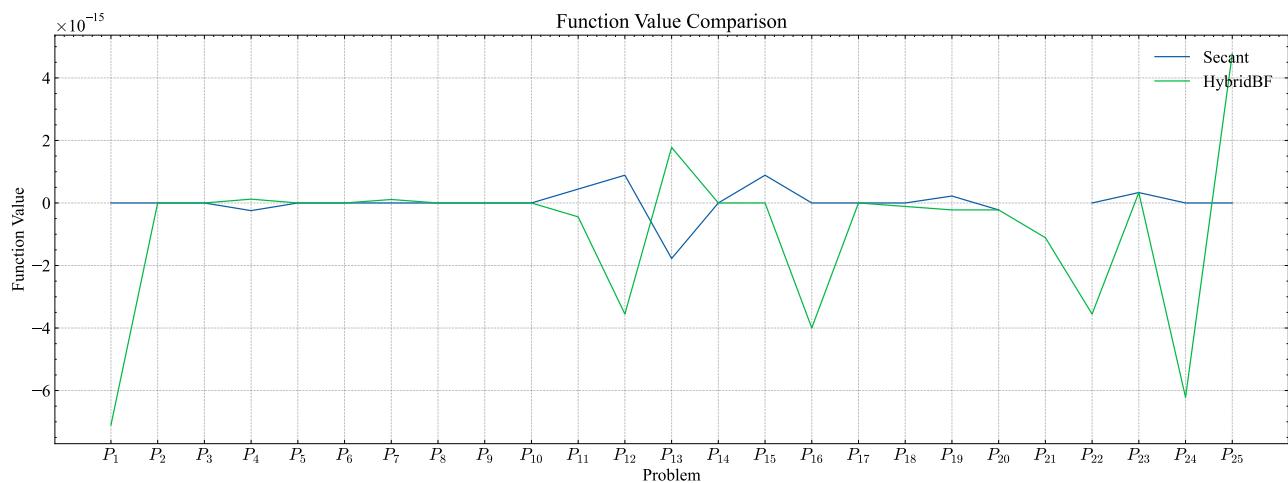


Figure 11: Function Value Comparison Secant, HybridBF

Development Environment:

- 🐧 Operating System: Linux Mint 21.3 Cinnamon
- 🐍 Python Version: 3.11.6
- ✎ Editor: Visual Studio Code 1.89.1
- Processor: Intel® Core™ i5-8300H CPU @ 2.30GHz × 4
- Memory: 16GB

### 3 Chapter Three: IBGA Algorithm

#### 3.1 Introduction

This chapter provides an overview of the proposed polynomial roots based encryption algorithm IBGA. The chapter contains the encryption and decryption processes, the algorithm's steps, and the results of the algorithm's performance compared to AES.

Note:

Unlike BiNew algorithm which is mentioned in the paper, this algorithm doesn't have coefficients rotation step.

#### 3.2 IBGA Algorithm Breakdown

##### 3.2.1 Encryption

The algorithm encrypts plaintext message using a polynomial and root finding method to generate a ciphertext. The encryption process works as follows:

1. Take the plaintext message and convert each 10 characters to an integer value using their ASCII values.
2. Take the key from the user which will be used to generate the polynomial. The encryption key consists of a set of 5 integer values  $x_1$ ,  $x_2$ ,  $y$ ,  $s$ , and  $r$ .
  1.  $x_1$  and  $x_2$  define  $x$  interval for the polynomial.
  2.  $y$  defines the start of  $y$  interval for the polynomial and the end will be the negative of  $y$  to ensure that the polynomial crosses the x-axis and has a root.
  3.  $s$  defines the number of sections that we want to divide the interval into which will affect the degree of the polynomial.
  4.  $r$  is used as a random state for the random number generator. The random values will always be the same for the same  $r$  value.
3. Use the encryption key to generate points that will be used to generate the polynomial. The points are generated using the following steps:
  1. Divide the interval  $[x_1, x_2]$  into  $s$  equal sections.
  2. Divide the interval  $[y, -y]$  into  $s$  equal sections.
  3. Generate points from the two intervals that will be used to generate the polynomial.
  4. Use the random numbers we got  $r$  to add some noise to the points.
  5. Apply Newton Forward Difference interpolation to the points to generate the polynomial Since the points are equally spaced.
4. Now we have the polynomial and the plaintext integer representation so we will normalize the plaintext integer representation to be between -1, 1 via this formula:

$$T' = \frac{T - \min}{\max - \min}$$

Where:  $\begin{cases} T & \text{is the original plaintext integer representation} \\ T' & \text{is the normalized plaintext integer representation} \\ \max & \text{is the maximum value of the plaintext integer representation} \\ \min & \text{is the minimum value of the plaintext integer representation} \end{cases}$  (5)

5. We get the root of the polynomial which will be the ciphertext using any of the root finding algorithm mentioned before.

6. Then we subtract the normalized plaintext integer value from the polynomial representation.

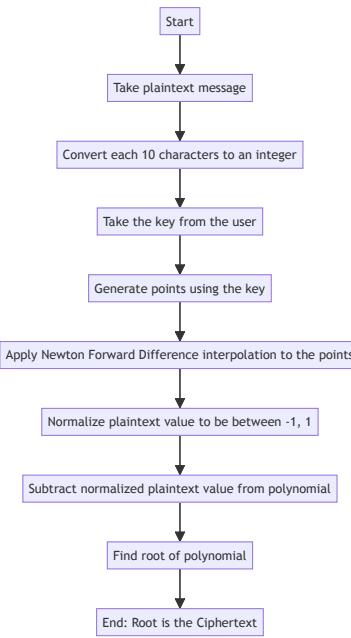


Figure 12: Encryption Steps Flowchart

### 3.2.2 Decryption

The algorithm decrypts the ciphertext message using the polynomial. The decryption process works as follows:

1. Take the ciphertext and the key from the user which will be used to generate the polynomial again.
2. Use the key to generate the polynomial using the same steps as the encryption process.
3. Now we have the polynomial and the ciphertext so we will substitute the ciphertext in the polynomial to get the normalized plaintext integer representation.
4. Get the original plaintext integer representation from the normalized plaintext integer representation using the following formula:

$$T = (T' \times (max - min)) + min \quad (6)$$

5. Convert the integer representation to the plaintext message by converting each integer to 10 characters using their ASCII values.

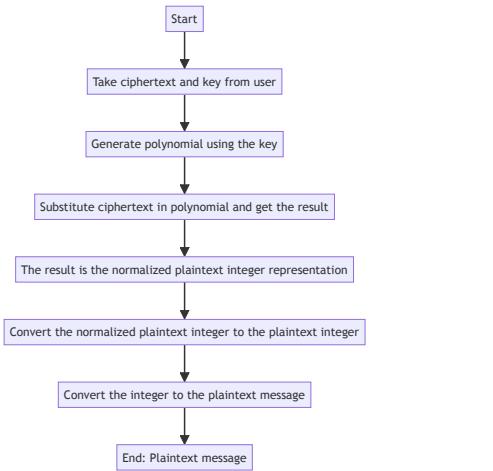


Figure 13: Decryption Steps Flowchart

### 3.3 Algorithm Pseudocode

#### 3.3.1 Encryption

These are the encryption steps of the IBGA algorithm. The encryption process involves converting each chunk in the plaintext into an integer, generating points using the key, interpolating a polynomial using the points, normalizing the integer plaintext chunk, and finding the ciphertext chunk which is the root of the polynomial we got after subtracting the normalized plaintext chunk from the polynomial. Finally, we append the ciphertext chunk to the ciphertext array.

---

#### Algorithm 1 IBGA Algorithm Encryption Steps

---

```

1: Input: plaintext, key
2: Output: ciphertext
3: procedure ENCRYPTION(plaintext, key)
4:    $x_1, x_2, y, s, r \leftarrow key$ 
5:   Initialize ciphertext as an empty array
6:   for each chunk in plaintext do                                 $\triangleright$  Each chunk is 10 characters
7:     integer_plaintext  $\leftarrow$  CONVERTTOINTEGER(chunk)
8:     points  $\leftarrow$  GENERATEPOINTS( $x_1, x_2, y, s, r$ )
9:     polynomial  $\leftarrow$  NEWTONINTERPOLATION(points)
10:    normalized_plaintext  $\leftarrow$  NORMALIZE(integer_plaintext)
11:    new_polynomial  $\leftarrow$  polynomial  $-$  normalized_plaintext
12:    ciphertext_chunk  $\leftarrow$  FINDROOT(new_polynomial)
13:    Append ciphertext_chunk to ciphertext array
14:   end for
15:   return ciphertext array
16: end procedure
    
```

---

#### 3.3.2 Decryption

These are the decryption steps of the IBGA algorithm. The decryption process is the reverse of the encryption process. We first generate the points using the key, then interpolate the polynomial using the points. We then substitute the ciphertext chunk into the polynomial to get the normalized plaintext chunk. We denormalize the normalized plaintext chunk to get the integer plaintext chunk. Finally, we convert the integer plaintext chunk to a string and append it to the plaintext string.

**Algorithm 2** IBGA Algorithm Decryption Steps

---

```

1: procedure DECRYPTION(ciphertext, key)
2:   Input: ciphertext, key
3:   Output: plaintext
4:    $x_1, x_2, y, s, r \leftarrow key$ 
5:   Initialize plaintext as an empty string
6:   for each ciphertext_chunk in ciphertext array do
7:      $\triangleright ciphertext\_chunk$  is the number we got from encrypting each text chunk
8:     points  $\leftarrow$  GENERATEPOINTS( $x_1, x_2, y, s, r$ )
9:     polynomial  $\leftarrow$  NEWTONINTERPOLATION(points)
10:    normalized_plaintext  $\leftarrow$  polynomial(ciphertext_chunk)
11:    integer_plaintext  $\leftarrow$  DENORMALIZE(normalized_plaintext)
12:    plaintext_chunk  $\leftarrow$  CONVERTTOCHARACTERS(integer_plaintext)
13:    Append plaintext_chunk to plaintext string
14:   end for
15:   return plaintext
16: end procedure

```

---

## 3.4 Results

We evaluated the performance of our proposed algorithm against established root finding algorithms. This evaluation compared CPU time across various text sizes and polynomial degrees in terms of encryption decryption and total times. Following this analysis, we selected the most efficient numerical method for further testing. Subsequently, we compared our algorithm's performance to the widely used AES symmetric encryption algorithm. The results demonstrated that our algorithm exhibits significantly faster execution times, particularly for larger text sizes.

### 3.4.1 With Different Root Finding Algorithms

We tested our algorithm with different root finding algorithms to determine the most efficient method. We compared the CPU time of the Bisection, False Position, HybridBF, Secant, and HybridSF methods. The results showed that the Secant method was the fastest for all encryption, decryption, and total times in terms of all text sizes and polynomial degrees and the Bisection method consistently exhibited the slowest performance across all evaluated text sizes and polynomial degrees.

The other methods showed nearly similar performance in terms of CPU time, with the HybridSF method demonstrating slightly faster execution times compared to other methods. The HybridBF method exhibited comparable performance to the False Position method across evaluated text sizes.

**3.4.1.1 Different Degrees** We compared the CPU time of the root finding algorithms for different polynomial degrees from 2 to 6. The results showed that the Secant method was the fastest for all degrees in terms of all encryption, decryption, and total times. The HybridSF method was the second fastest and the Bisection method consistently exhibited the slowest performance across all polynomial degrees.

**3.4.1.1.1 Encryption Time** As illustrated in the plot, the secant method (depicted by the dotted line) emerged as the most efficient approach in terms of CPU time required for the encryption process. Notably, all other evaluated methods exhibited slower execution times. The bisection method, in particular, demonstrated significantly lower performance compared to the remaining algorithms.

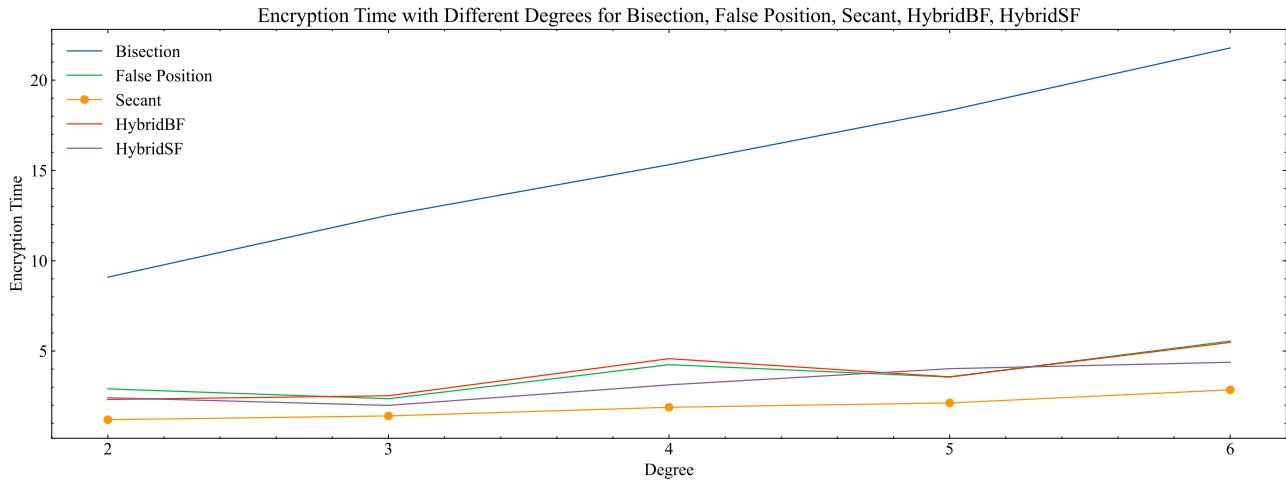


Figure 14: Root Finding Algorithms Encryption Time With Different Degrees

**3.4.1.1.2 Decryption Time** The decryption process in our algorithm involves direct polynomial evaluation using the ciphertext. Consequently, the choice of the root-finding algorithm employed during encryption has no effect on decryption time. This explains the observed consistency in decryption times across the various root-finding methods.

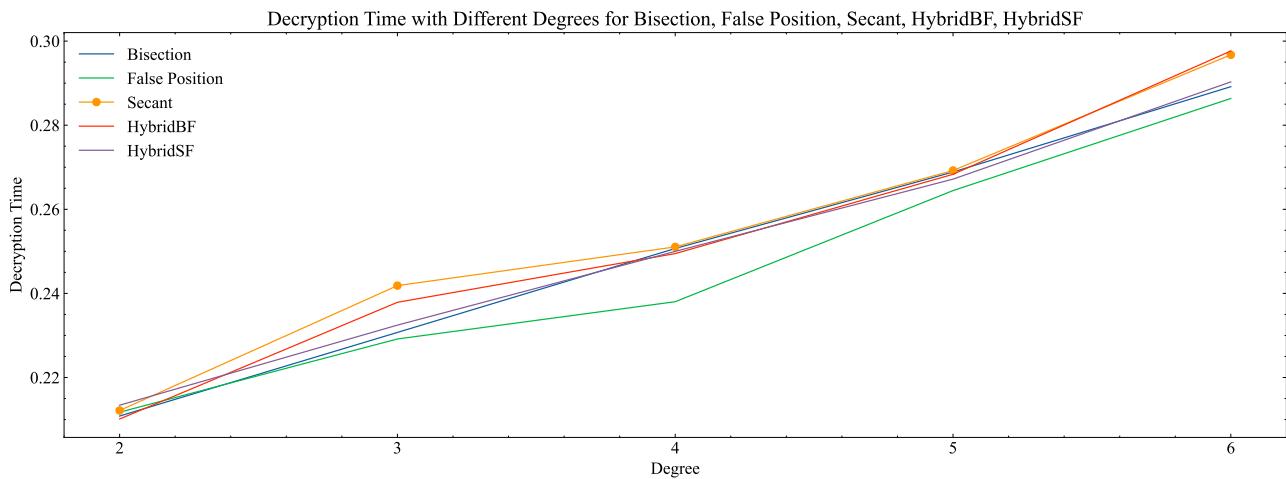


Figure 15: Root Finding Algorithms Decryption Time With Different Degrees

**3.4.1.1.3 Total Time** The total time required for both encryption and decryption processes is directly influenced by the root-finding algorithm employed during encryption. As depicted in the plot, the secant method consistently achieved the fastest processing times across all evaluated polynomial degrees. Conversely, the bisection method exhibited the slowest performance throughout the experiments. This trend underlines the critical role of selecting an efficient root-finding algorithm to optimize the overall execution time of the encryption and decryption processes.

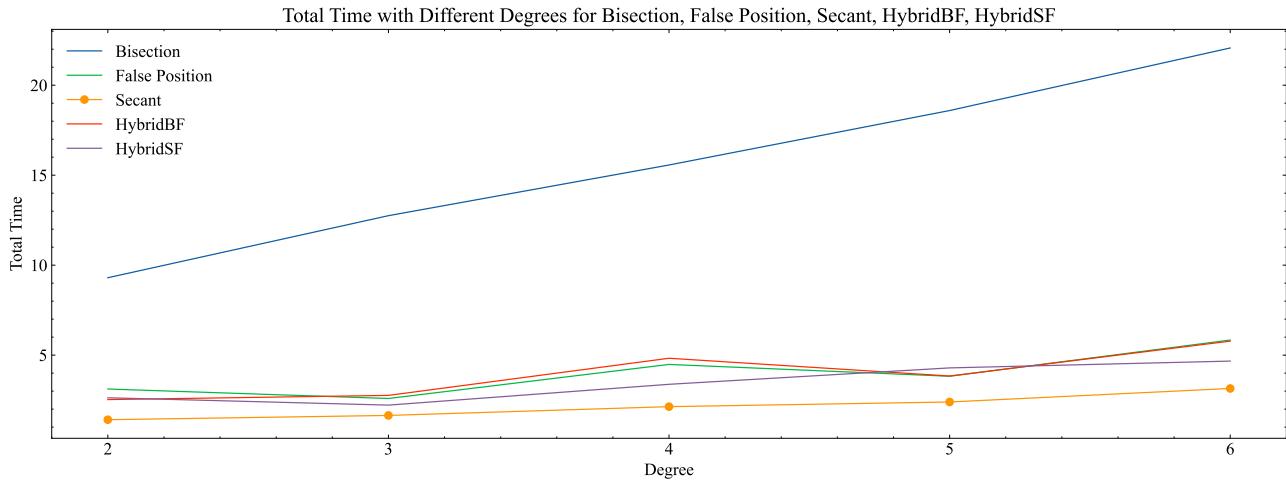


Figure 16: Root Finding Algorithms Total Time With Different Degrees

**3.4.1.2 Different File sizes** We also evaluated the performance of the root-finding algorithms across varying text sizes. The results demonstrated that the secant method consistently exhibited the fastest execution times for all file sizes. Conversely, the bisection method consistently demonstrated the slowest performance across all evaluated file sizes.

**3.4.1.2.1 Encryption Time** The secant method consistently outperformed all other root-finding algorithms in terms of encryption time across all evaluated file sizes. The bisection method exhibited the slowest performance, with significantly higher CPU times compared to the remaining algorithms.

Furthermore, the analysis identified that the HybridBF method displayed identical performance to the False Position method for all file sizes. The HybridSF method demonstrated slightly slower execution times compared to both HybridBF and False Position methods.

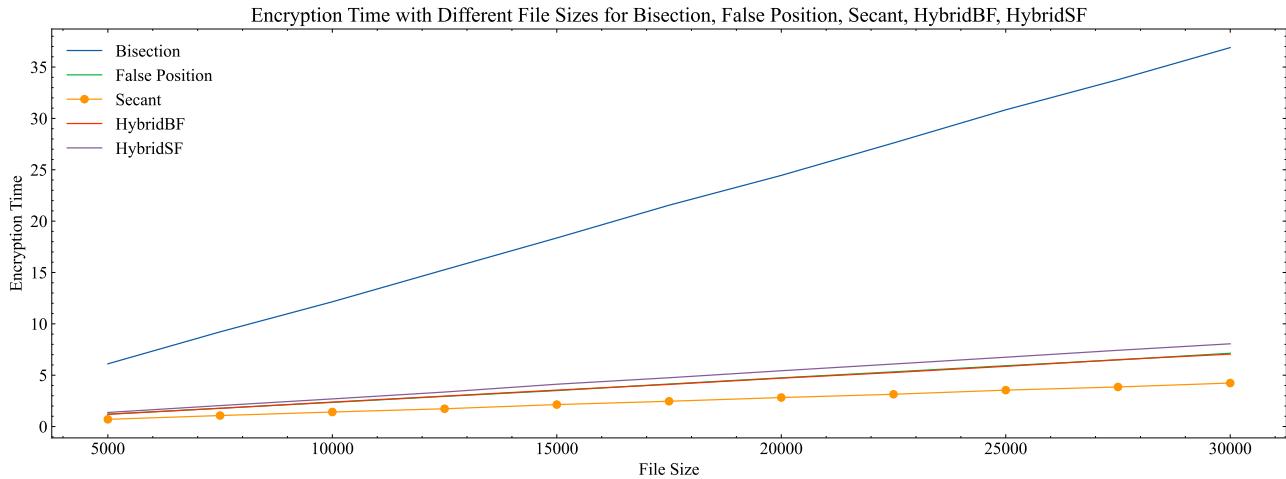


Figure 17: Root Finding Algorithms Encryption Time With Different File Sizes

**3.4.1.2.2 Decryption Time** The decryption process in our algorithm involves direct polynomial evaluation using the ciphertext. Consequently, the choice of the root-finding algorithm employed during encryption has no effect on decryption time. This explains the observed consistency in decryption times across the various root-finding methods.

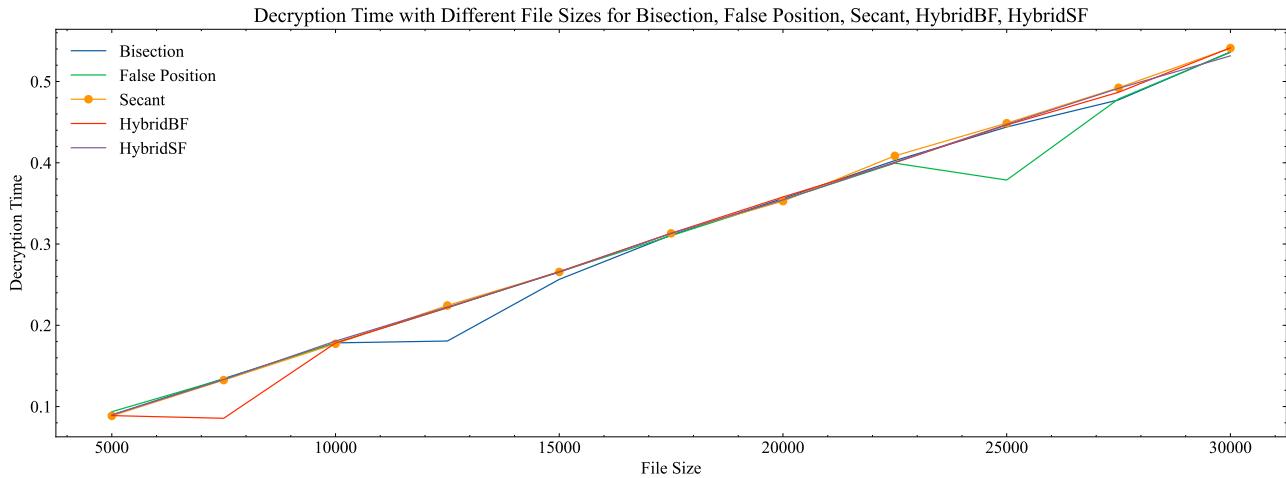


Figure 18: Root Finding Algorithms Decryption Time With Different File Sizes

**3.4.1.2.3 Total Time** The total time required for both encryption and decryption processes is directly influenced by the root-finding algorithm employed during encryption. As depicted in the plot, the secant method consistently achieved the fastest processing times across all evaluated file sizes. Conversely, the bisection method exhibited the slowest performance throughout the experiments. This trend underlines the critical role of selecting an efficient root-finding algorithm to optimize the overall execution time of the encryption and decryption processes.

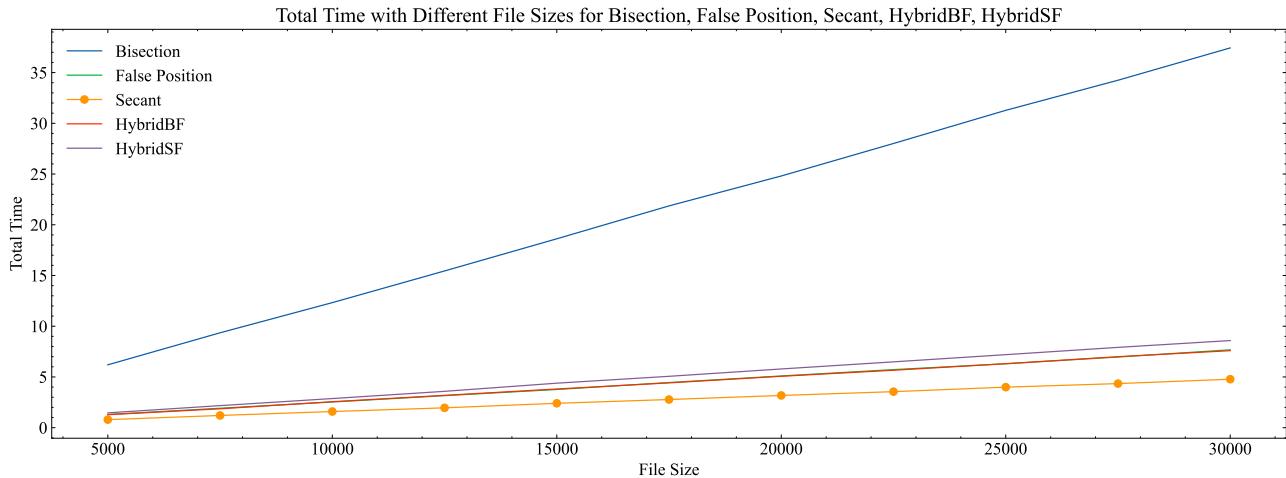


Figure 19: Root Finding Algorithms Total Time With Different File Sizes

### 3.4.2 Against AES

We compared the performance of our proposed algorithm against the widely used AES symmetric encryption algorithm. The results demonstrated that our algorithm exhibited significantly faster execution times for all evaluated text sizes.

**3.4.2.1 Different Text Sizes** We evaluated the performance of our algorithm against AES for different text sizes ranging from 5000 to 30000 bytes which is nearly equivalent to 50000 to 300000 characters. The results showed that our algorithm outperformed AES in terms of CPU time for decryption, and total times across all evaluated text sizes, but it was slower in terms of encryption time.

**3.4.2.1.1 Encryption Time** The plot illustrates the encryption time comparison between our algorithm and AES for different text sizes. Our algorithm had slower execution times compared to AES for all evaluated text sizes. This is attributed to the complexity of the encryption process in our algorithm, which involves polynomial interpolation and root finding. In contrast, the encryption process in AES is more straightforward and less computationally intensive.

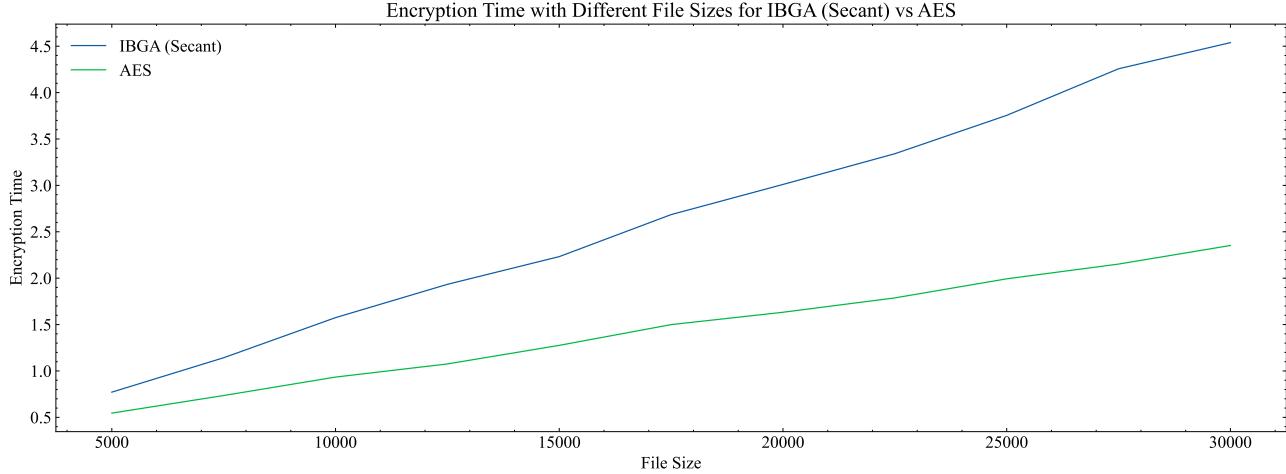


Figure 20: Encryption Time Comparison

**3.4.2.1.2 Decryption Time** The decryption time comparison between our algorithm and AES revealed that our algorithm consistently outperformed AES for all evaluated text sizes. This superior performance can be attributed to the direct polynomial evaluation process employed during decryption, which is more efficient compared to the decryption process in AES.

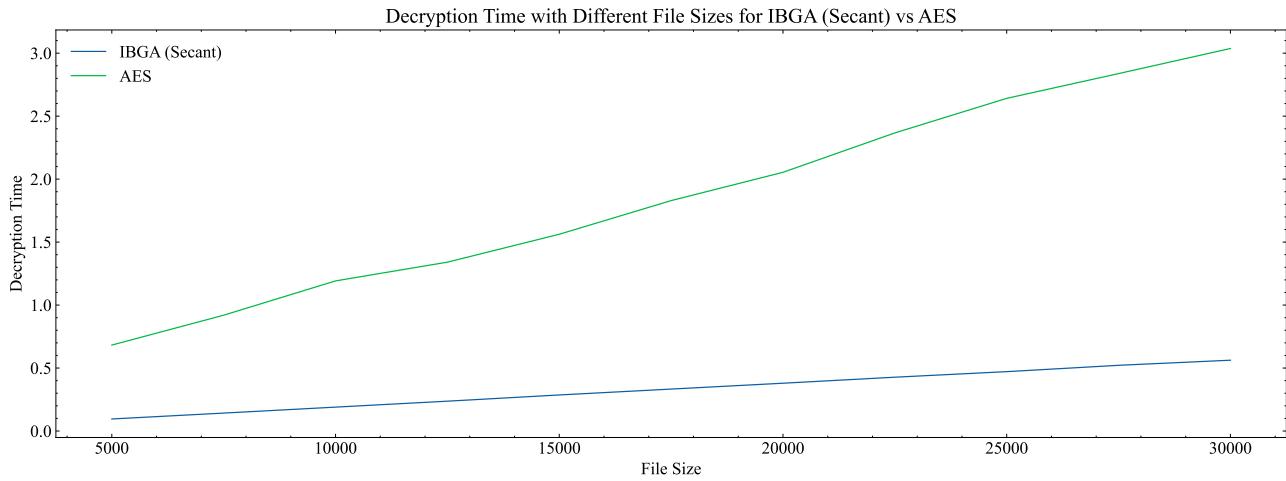


Figure 21: Decryption Time Comparison

**3.4.2.1.3 Total Time** The total time comparison between our algorithm and AES demonstrated that our algorithm exhibited faster execution times for all evaluated text sizes.

This superior performance is attributed to the efficient decryption process in our algorithm, which directly evaluates the polynomial using the ciphertext. In contrast, the decryption process in AES is more complex and computationally intensive, resulting in slower execution times.

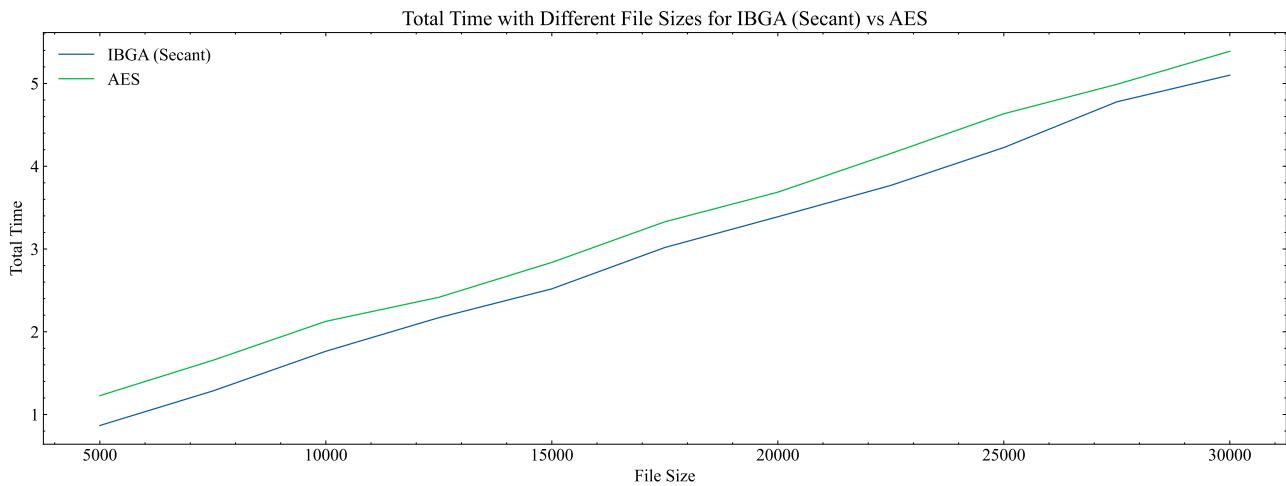


Figure 22: Total Time Comparison

**3.4.2.2 Different Polynomial Degrees** We evaluated the performance of our algorithm against AES. We used different polynomial degrees from 2 to 6 in our algorithm and different key sizes in AES from 12 characters to 60 characters.

The results showed that the AES performance was consistent across all evaluated key sizes. In contrast, our algorithm exhibited faster execution times for lower polynomial degrees and slower execution times for higher polynomial degrees. This trend can be attributed to the increased computational complexity associated with higher polynomial degrees.

**3.4.2.2.1 Encryption Time** The plot illustrates the encryption time comparison between our algorithm and AES for different polynomial degrees and key sizes. Our algorithm demonstrated slower execution times compared to AES for all evaluated polynomial degrees and the key sizes. This is attributed to the increased computational complexity associated with higher polynomial degrees in our algorithm.

We can also notice that the encryption time for our algorithm increases as the polynomial degree increases. This is due to the increased computational complexity associated with higher polynomial degrees. In contrast, the encryption time for AES remains consistent across all evaluated key sizes.

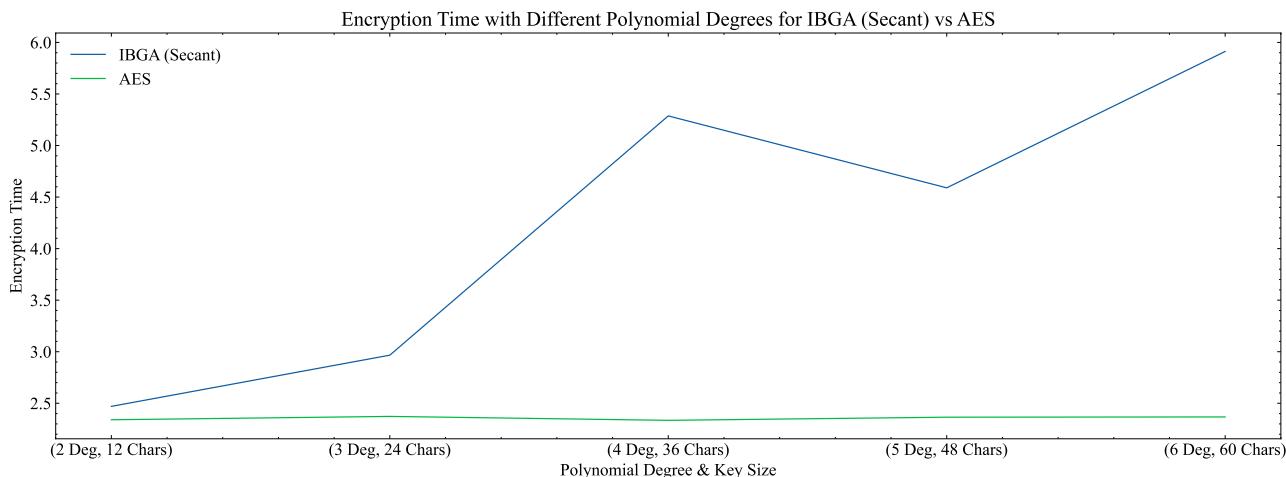


Figure 23: Encryption Time Comparison

**3.4.2.2.2 Decryption Time** The decryption time comparison between our algorithm and AES revealed that our algorithm consistently outperformed AES for all evaluated polynomial degrees and key sizes. This superior performance can be attributed to the direct polynomial evaluation process employed during decryption, which is more efficient compared to the decryption process in AES.

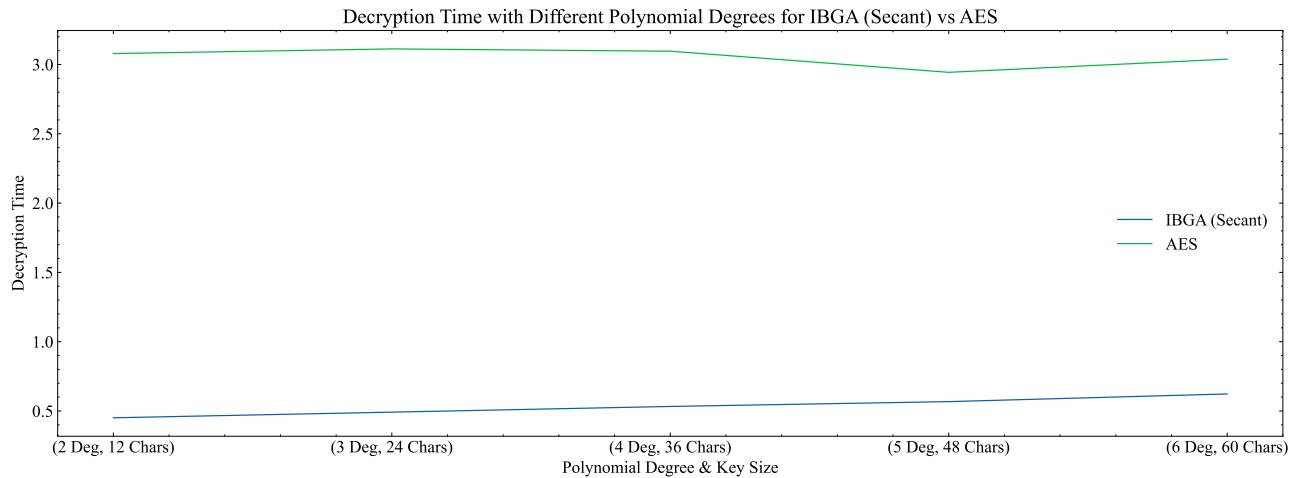


Figure 24: Decryption Time Comparison

**3.4.2.2.3 Total Time** The total time comparison between our algorithm and AES demonstrated that our algorithm exhibited faster execution times in the lower polynomial degrees and slower execution times in the higher polynomial degrees.

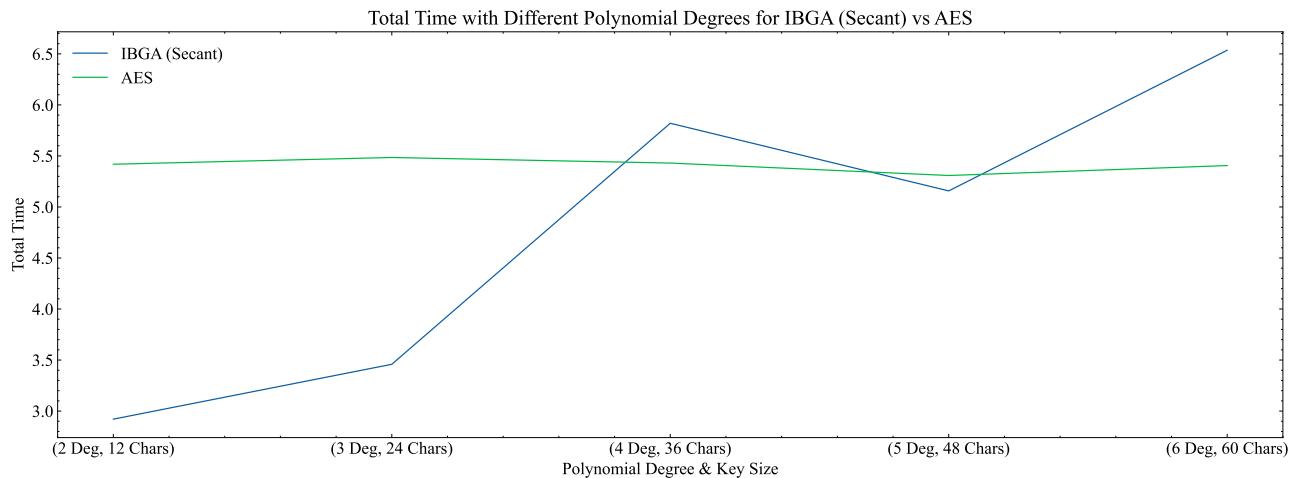


Figure 25: Total Time Comparison

### 3.4.3 Final Thoughts

The results of our algorithm evaluation demonstrated that the IBGA algorithm exhibits significantly faster execution times compared to AES for both encryption and decryption processes. This superior performance can be attributed to the direct polynomial evaluation process employed during decryption, which is more efficient compared to the decryption process in AES.

Development Environment:

We have used the same development environment as mentioned in the previous chapter.

## 4 Chapter Four: Design and Analysis

### 4.1 Introduction

The analysis and design section provides an overview of the proposed system architecture and data flow of the secure messaging web application. This section contains several software engineering diagrams depicting the structural and behavioral models of the system.

To enable secure communication, the application utilizes the cryptographic algorithm based on polynomial roots.

### 4.2 Functional & Non-Functional Requirements Table

Table 8: Functional & NonFunctional Requirements

Name	Functional	Non Functional	Description	Priority	Actor
Registration	✓		Functionality to create account.	High	User & Admin
Login	✓		Functionality to get access.	High	User & Admin
Logout	✓		Functionality to delete session.	High	User & Admin
Add Chat	✓		Functionality for user to chat with other.	Medium	User
Remove Chat	✓		Functionality for user to remove chat.	Medium	User
Find Chat	✓		Functionality for user to search for a chat.	High	User
Block Account	✓		Functionality for user to block user account.	Medium	User
Send Message	✓		Functionality that send message.	High	User
Delete Message	✓		Functionality to delete message.	Medium	User & Admin
Read Message	✓		Functionality for user to read message	High	User
Privacy		✓	Users privacy.	High	Admin
Robustness		✓	to make able to deal with errors.	High	Admin
Performance		✓	Application performance must be better.	High	Admin
Usability		✓	To make able to use even for newbie.	High	Admin
Reliability		✓	To be trustworthy to users.	High	Admin
Supportability		✓	To be capable of being supportive.	High	Admin

Name	Functional	Non Functional	Description	Priority	Actor
Portability		✓	Application must be able to run in many different system.	High	Admin

## 4.3 Diagrams

### 4.3.1 Use Case Diagram

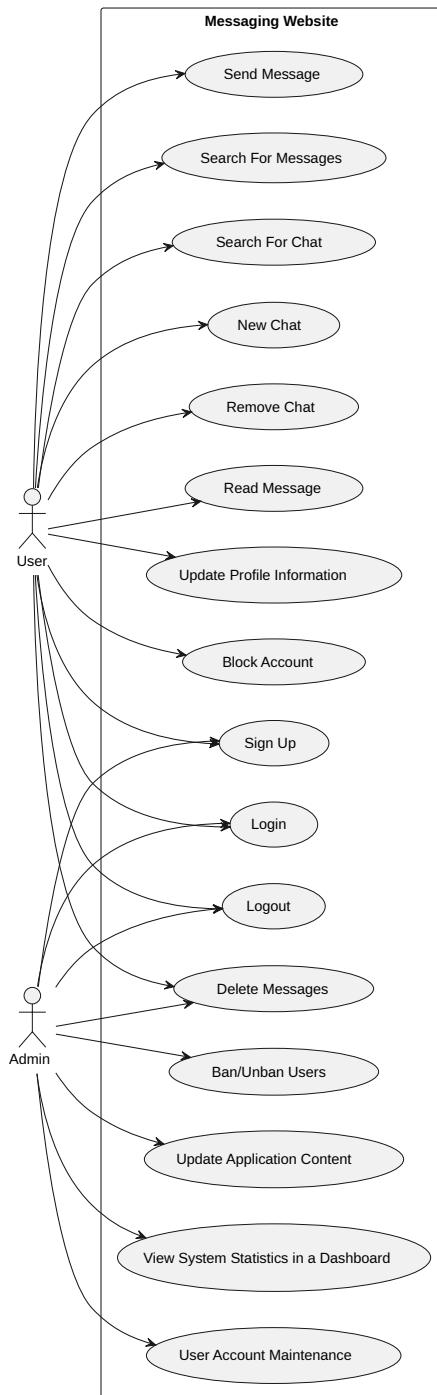


Figure 26: Use Case Diagram

#### 4.3.2 Activity Diagram

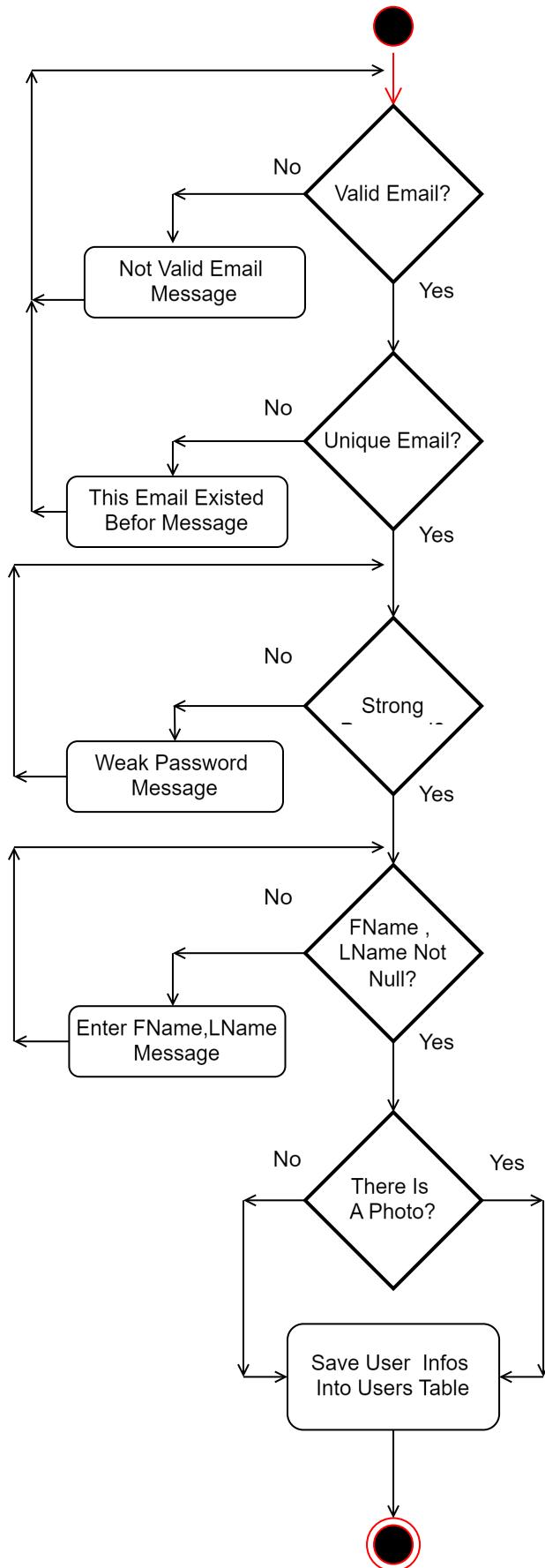


Figure 27: Sign up Activity Diagram

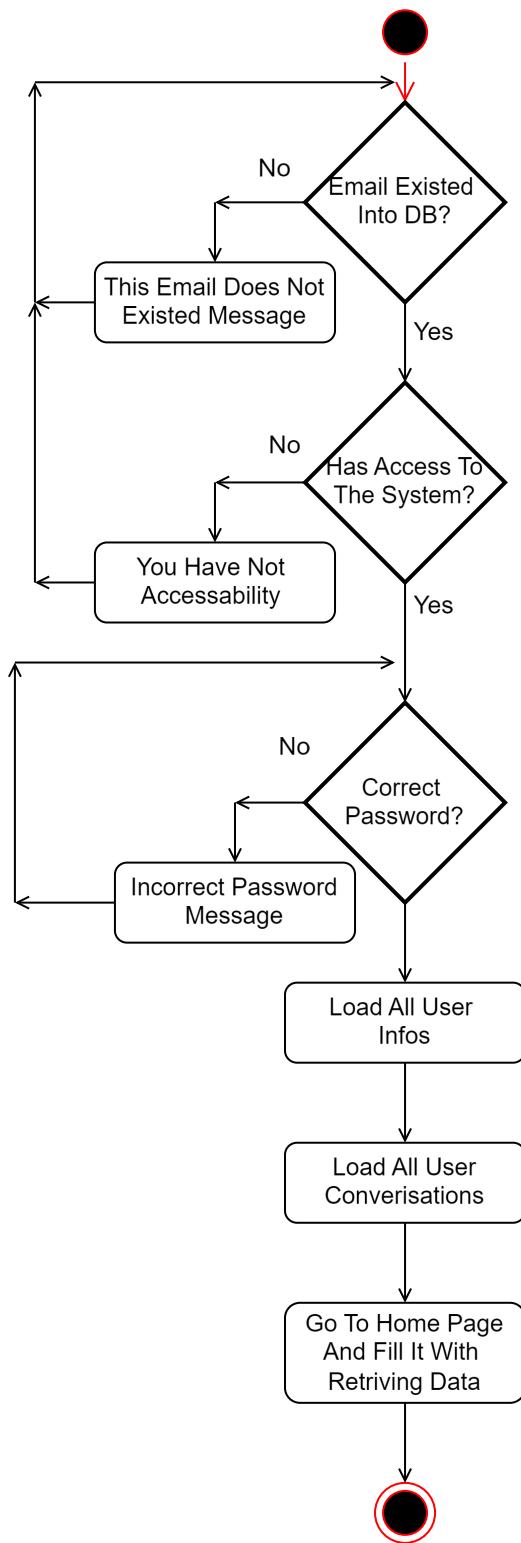


Figure 28: Sign in Activity Diagram

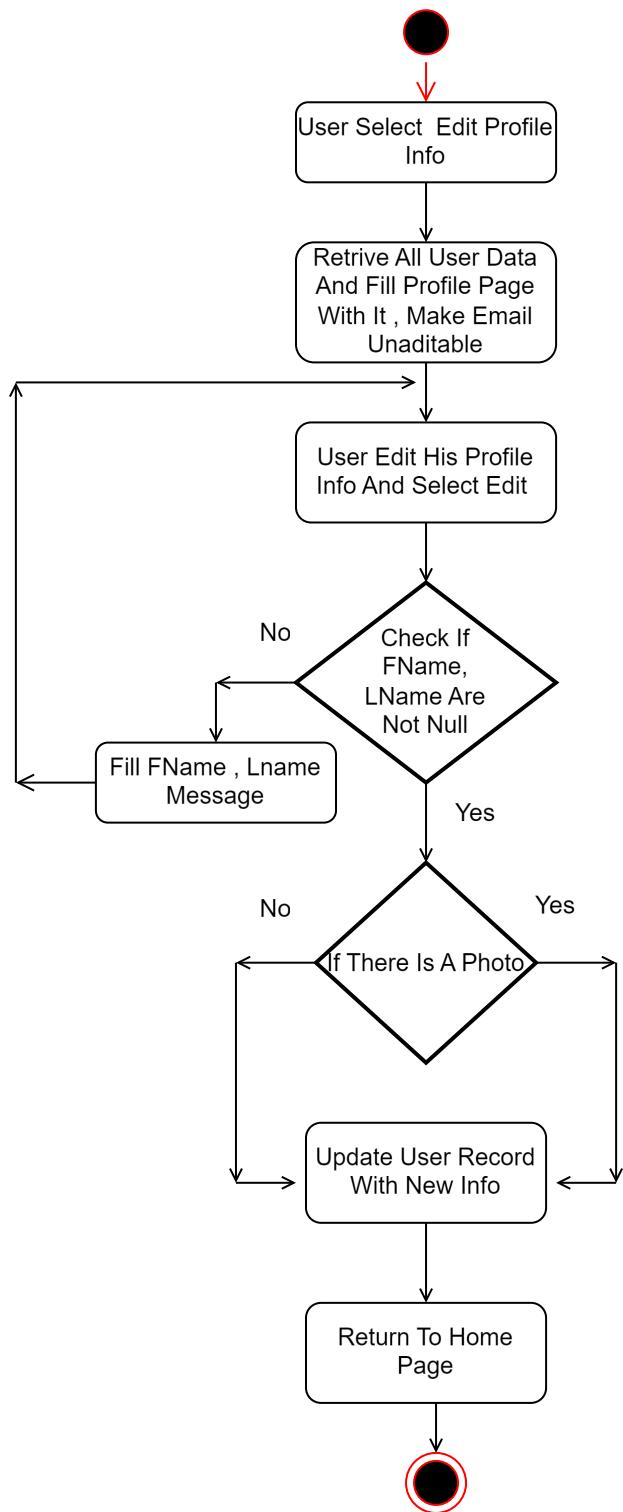


Figure 29: Edit Profile Activity Diagram

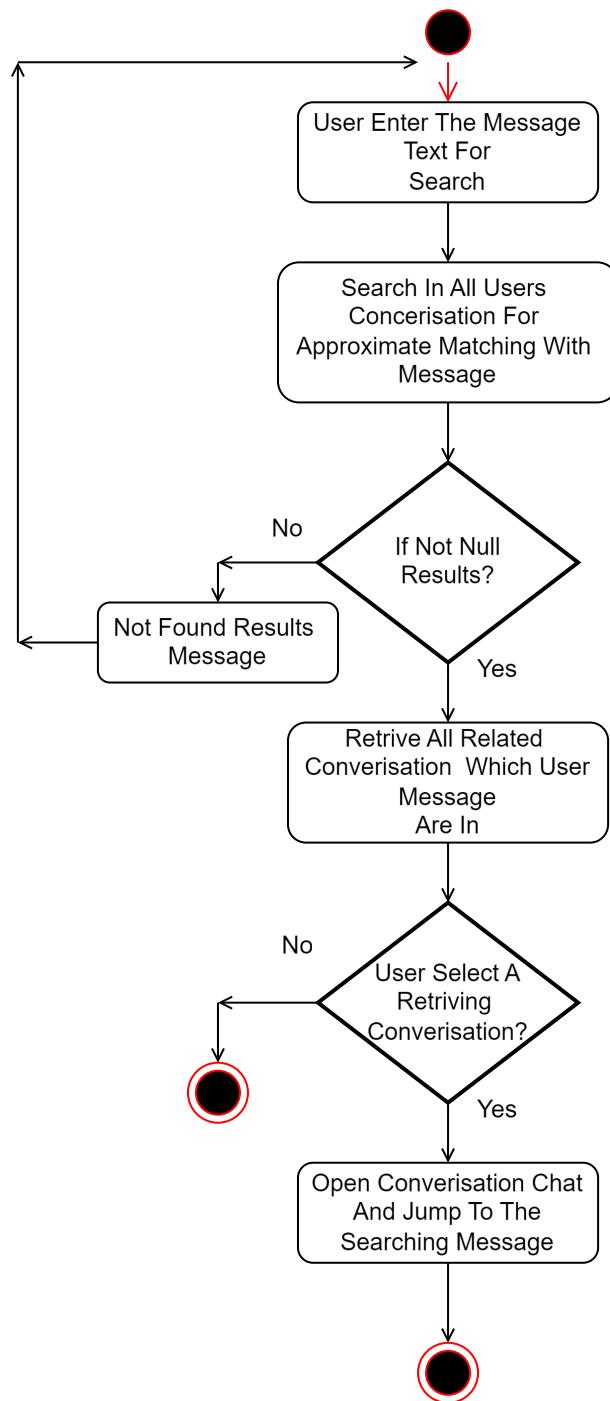


Figure 30: Search Message Activity Diagram

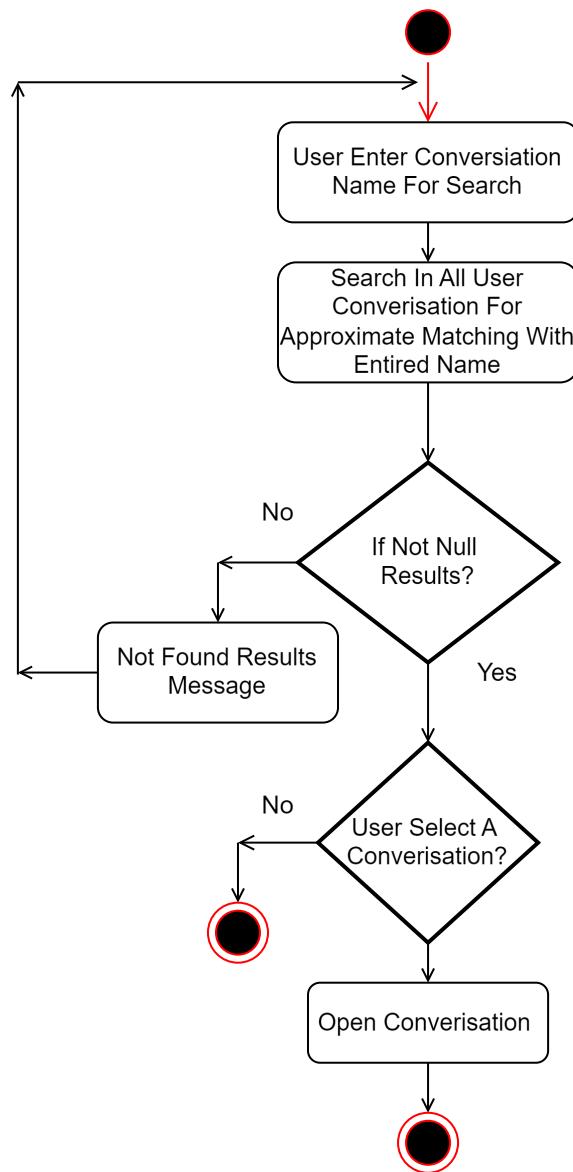


Figure 31: Search For Conversation Activity Diagram

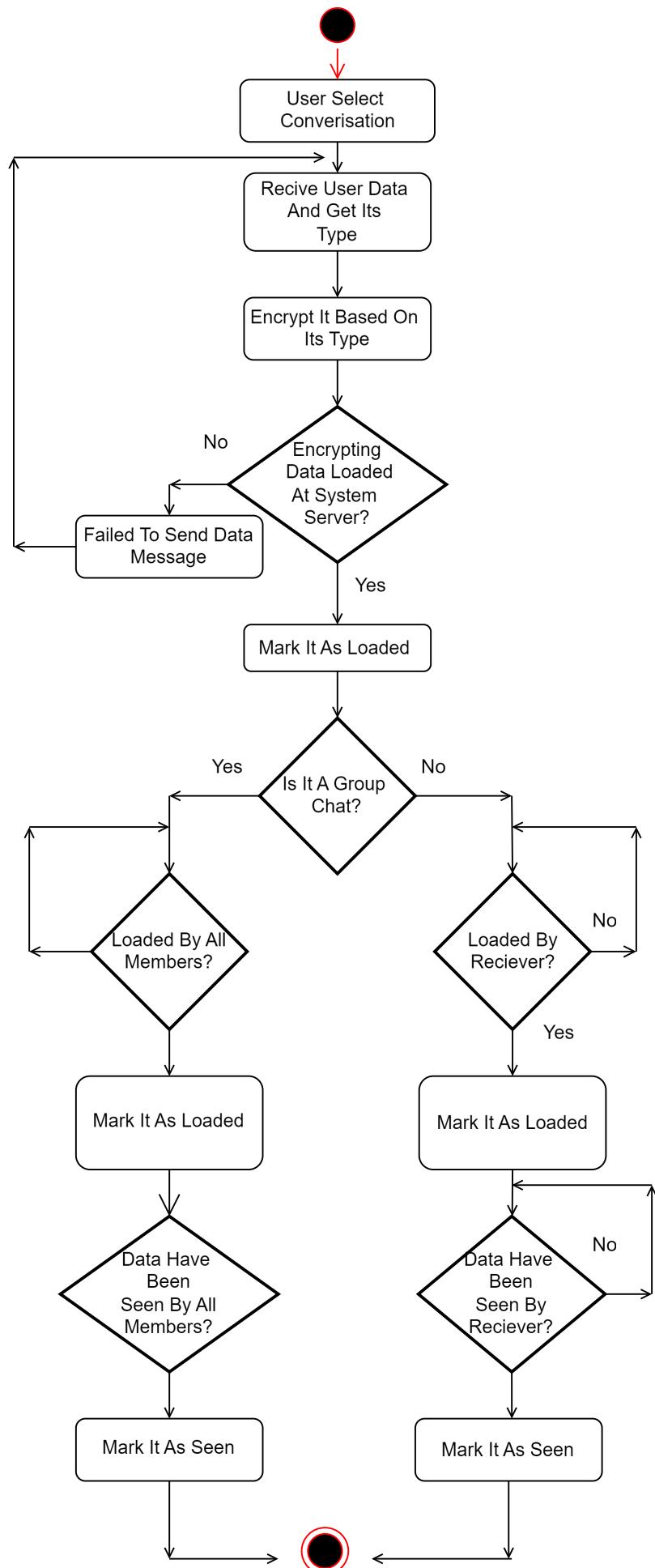


Figure 32: Search/Receive Data Activity Diagram

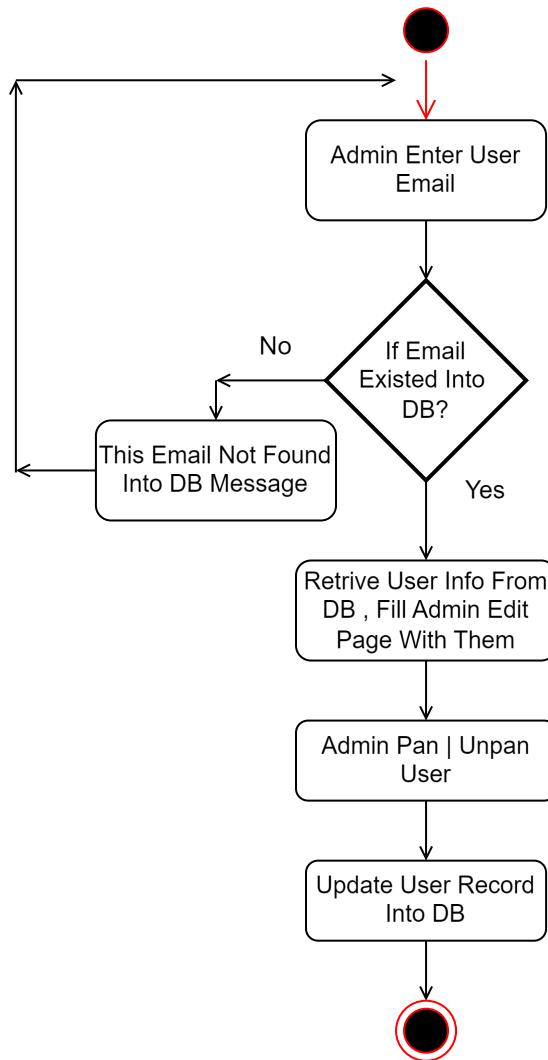


Figure 33: Ban Unban Users Activity Diagram

#### 4.3.3 Class Diagram

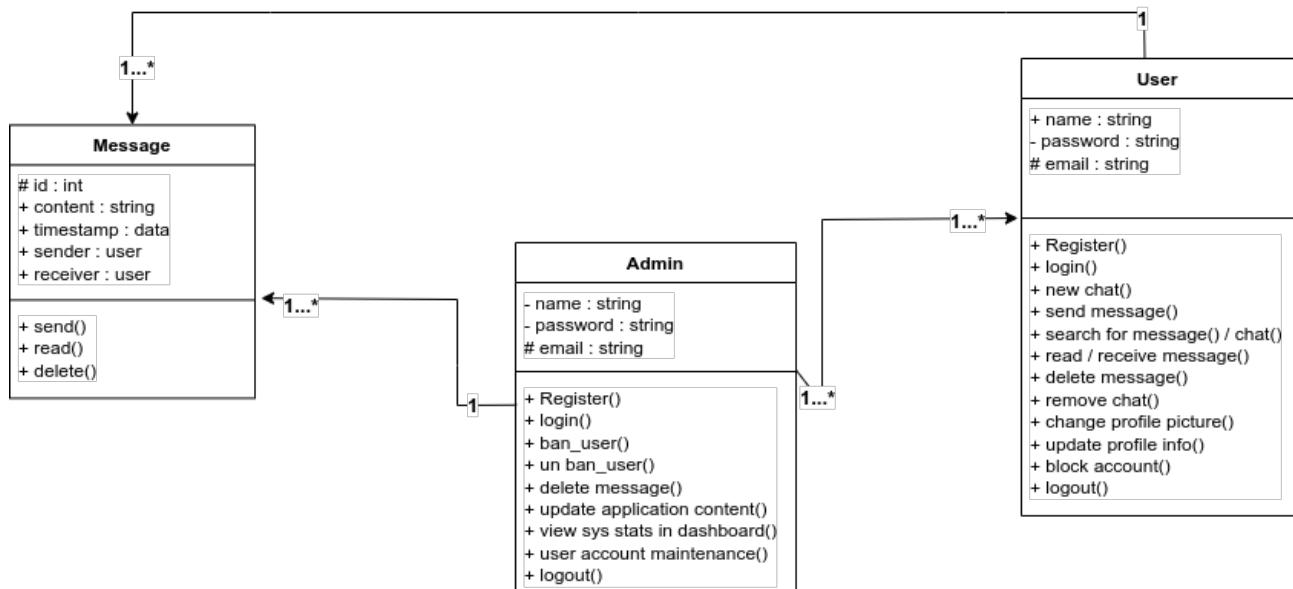


Figure 34: Class Diagram

#### 4.3.4 Sequence Diagram

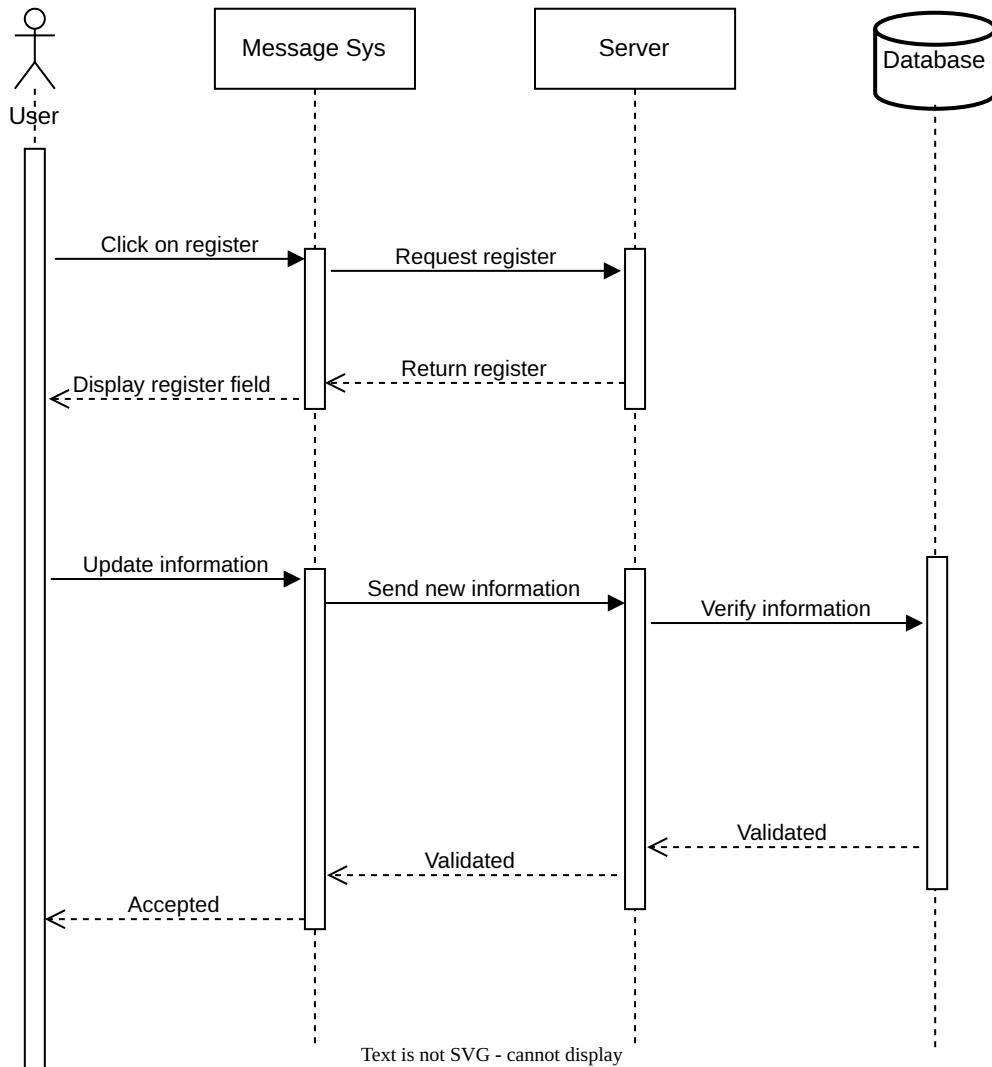


Figure 35: Registration Sequence Diagram

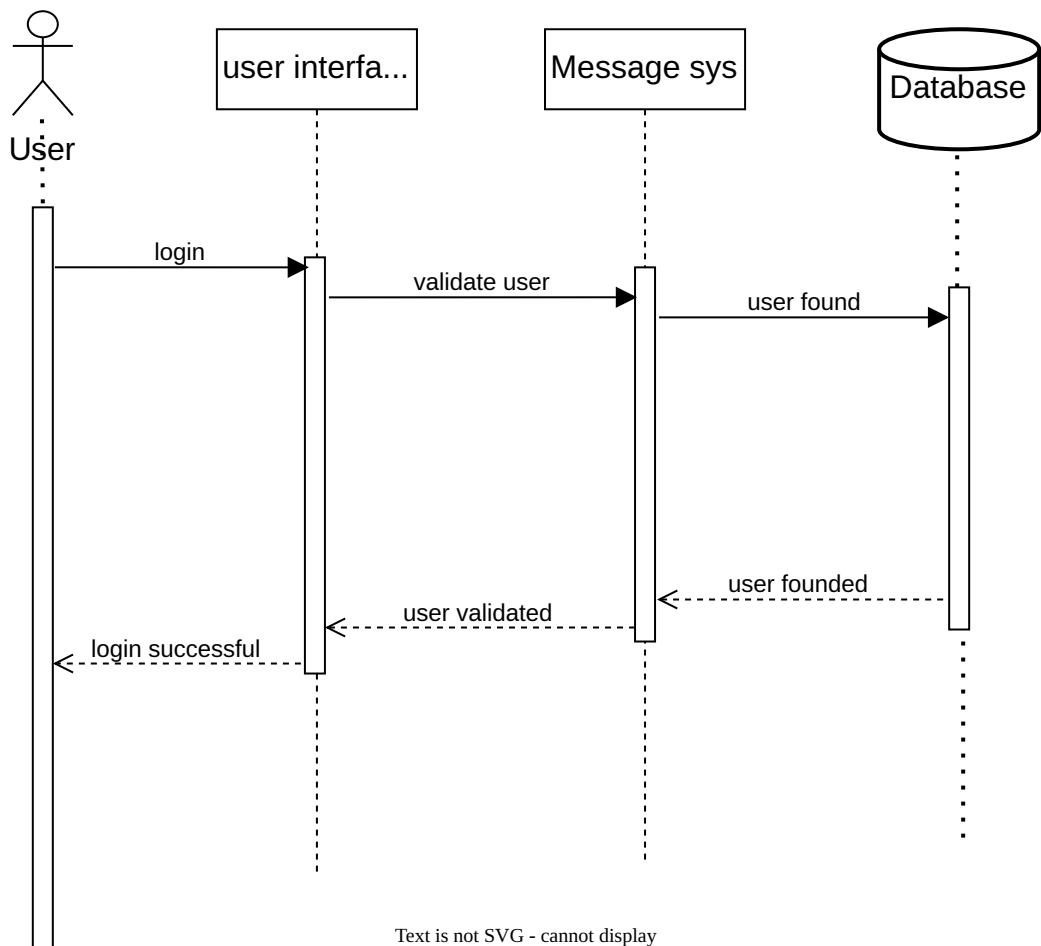


Figure 36: Login Sequence Diagram

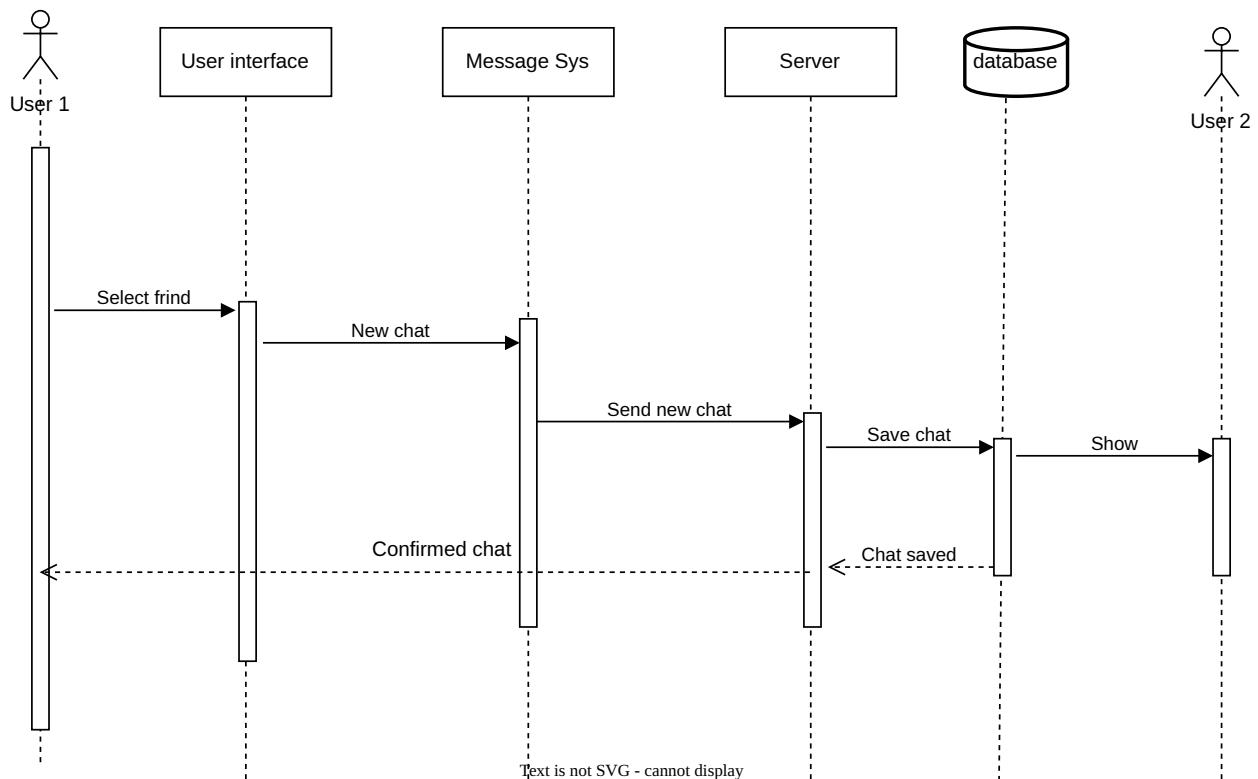


Figure 37: New Chat Sequence Diagram

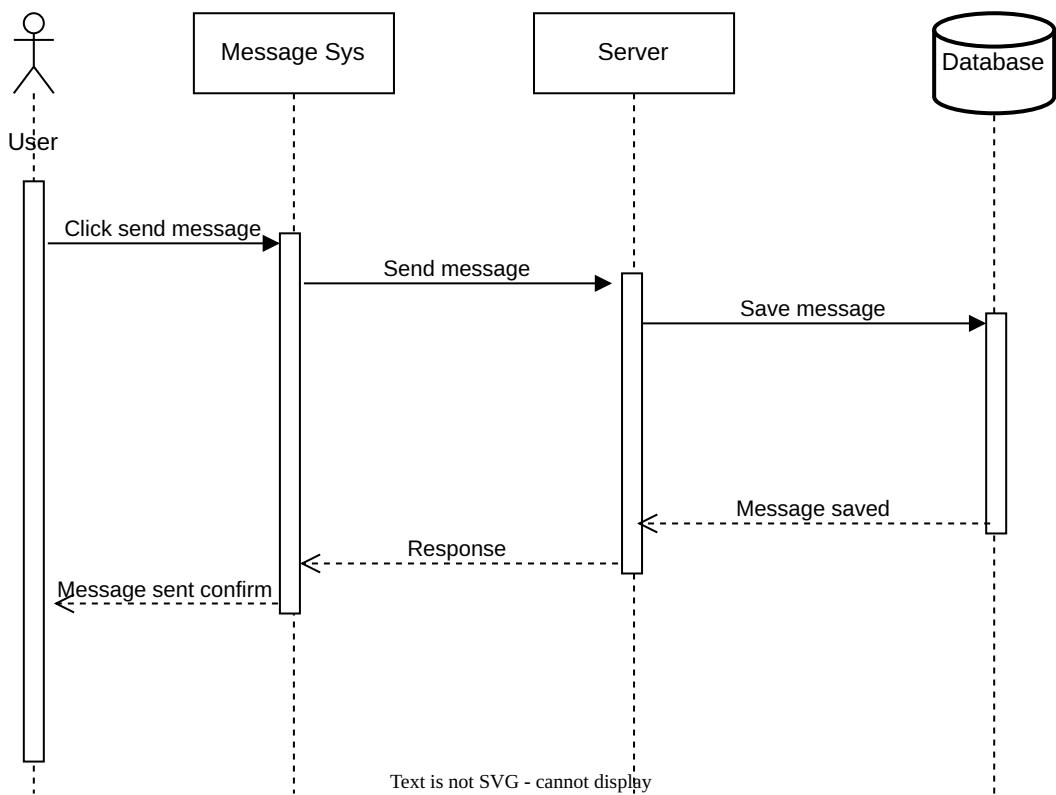


Figure 38: Send Message Sequence Diagram

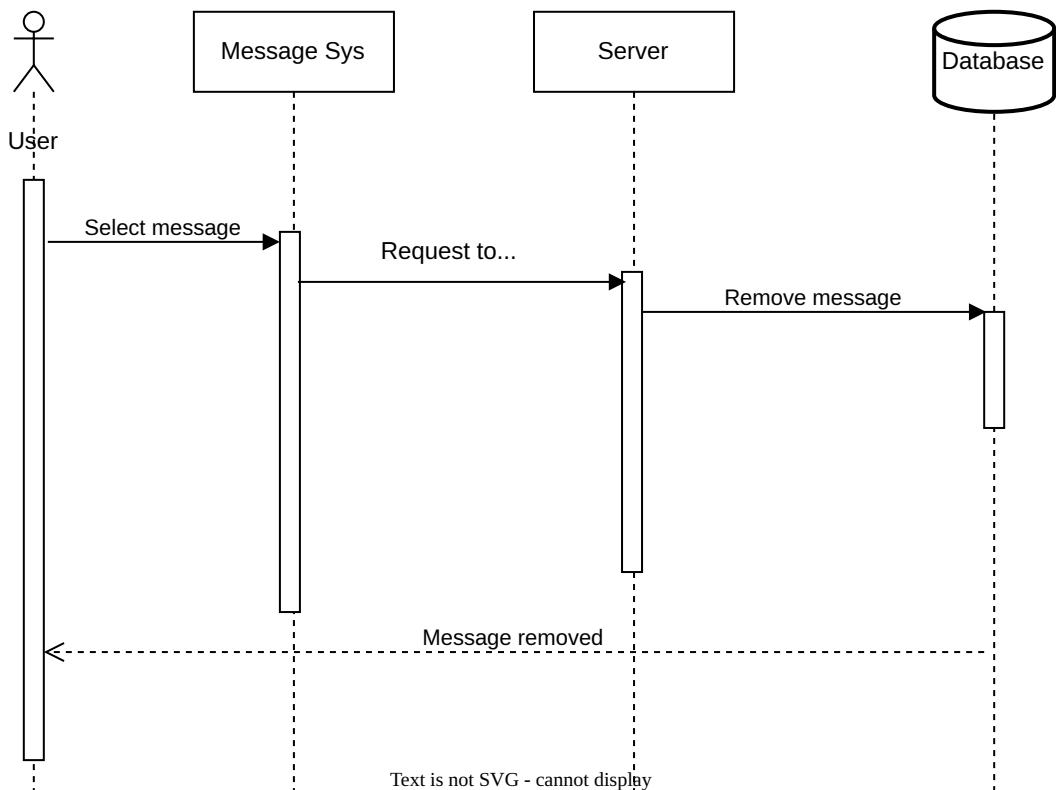


Figure 39: Remove Message Sequence Diagram

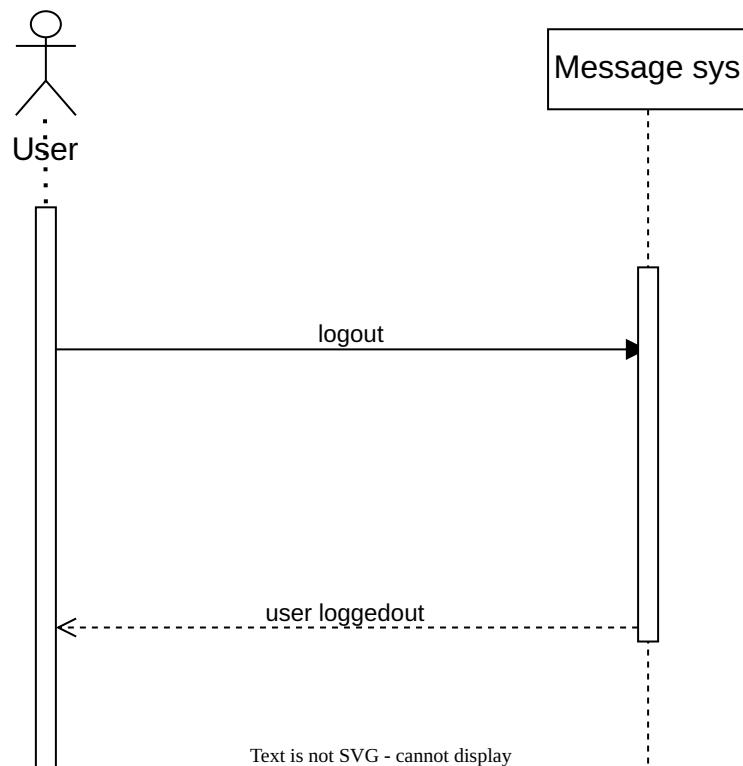


Figure 40: Logout Sequence Diagram

#### 4.3.5 State Diagram

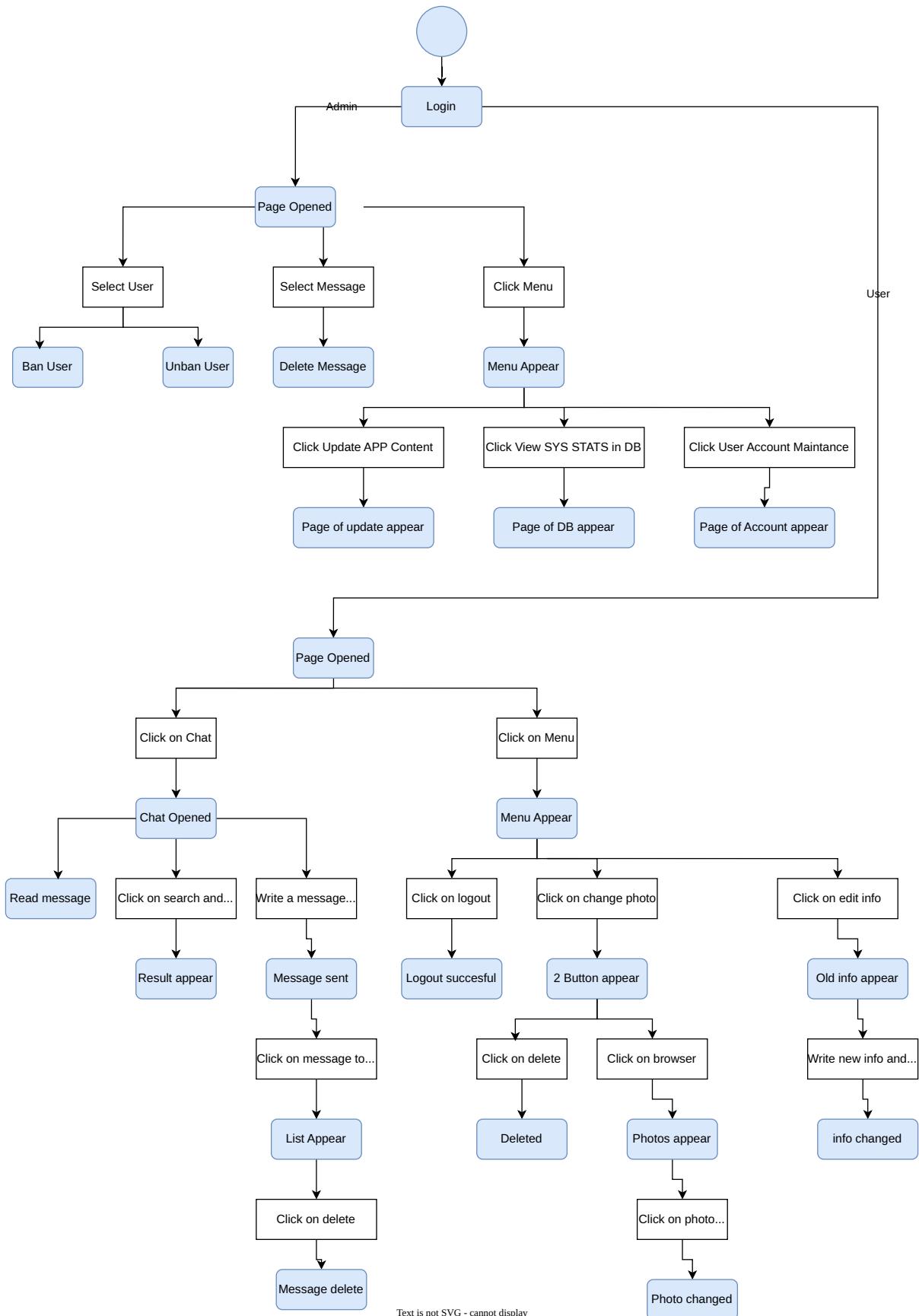


Figure 41: State Diagram

#### 4.3.6 Context Diagram

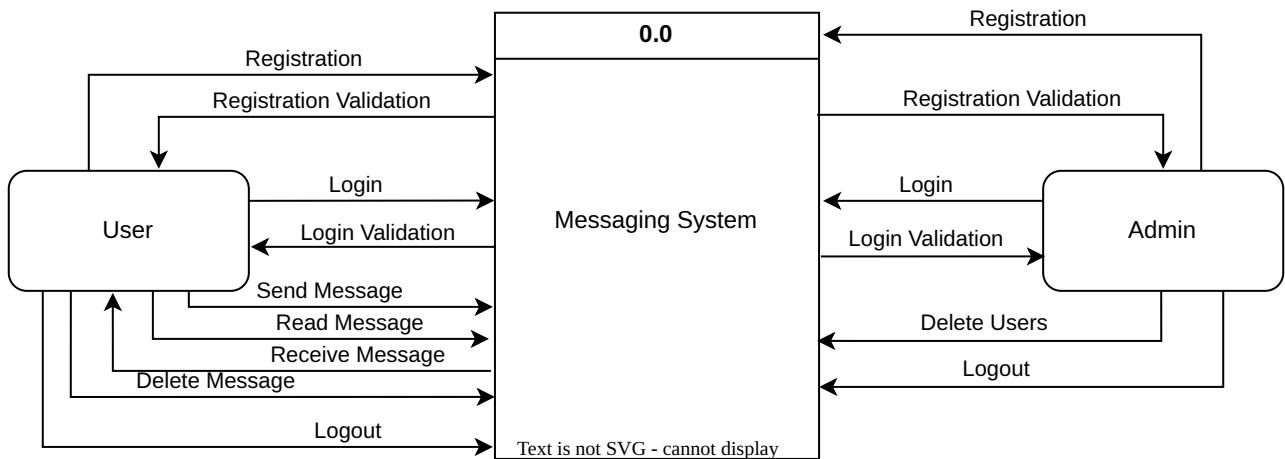


Figure 42: Context Diagram

#### 4.3.7 Data Flow Diagram

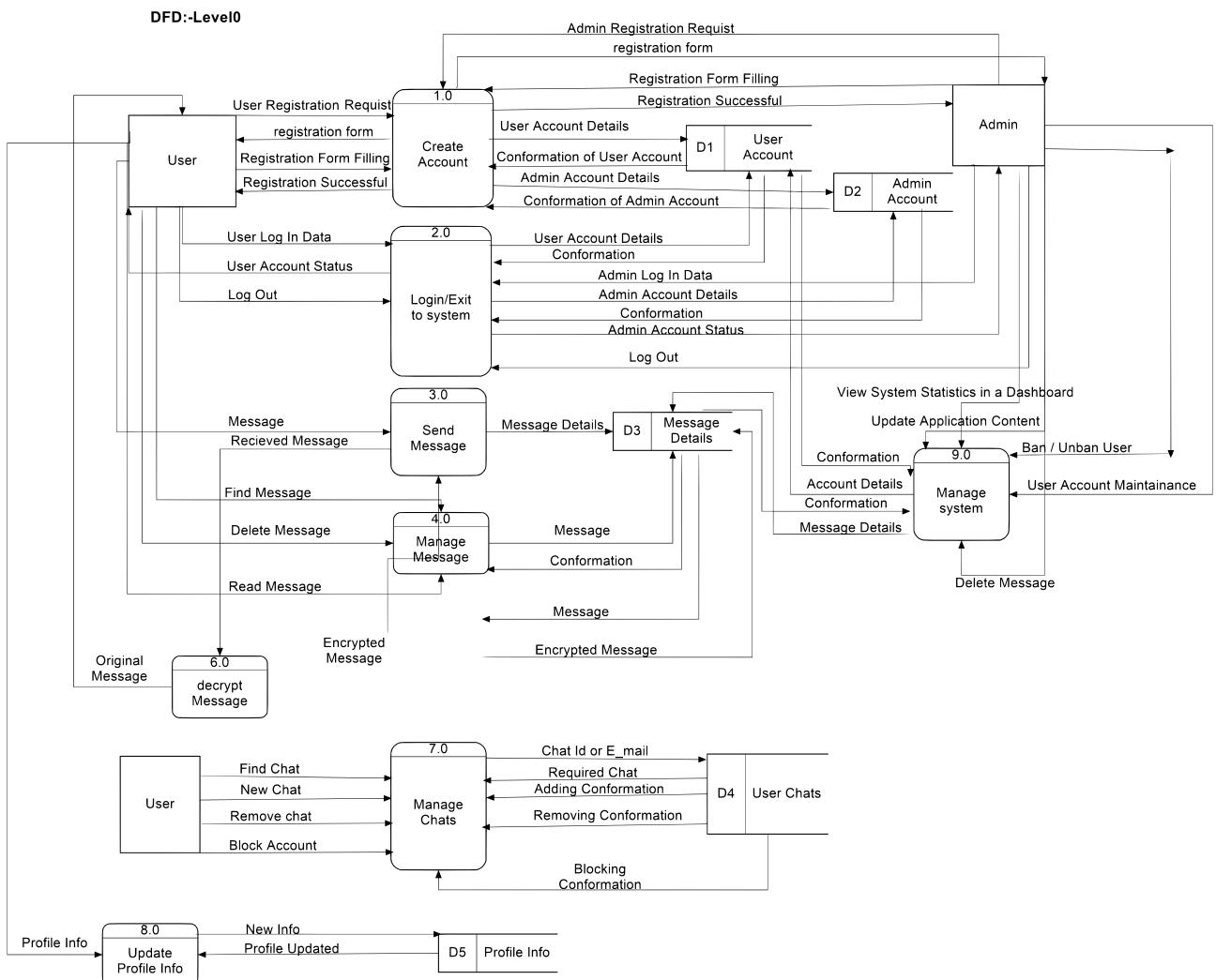


Figure 43: Data Flow Diagram

### 4.3.8 Entity Relationship Diagram

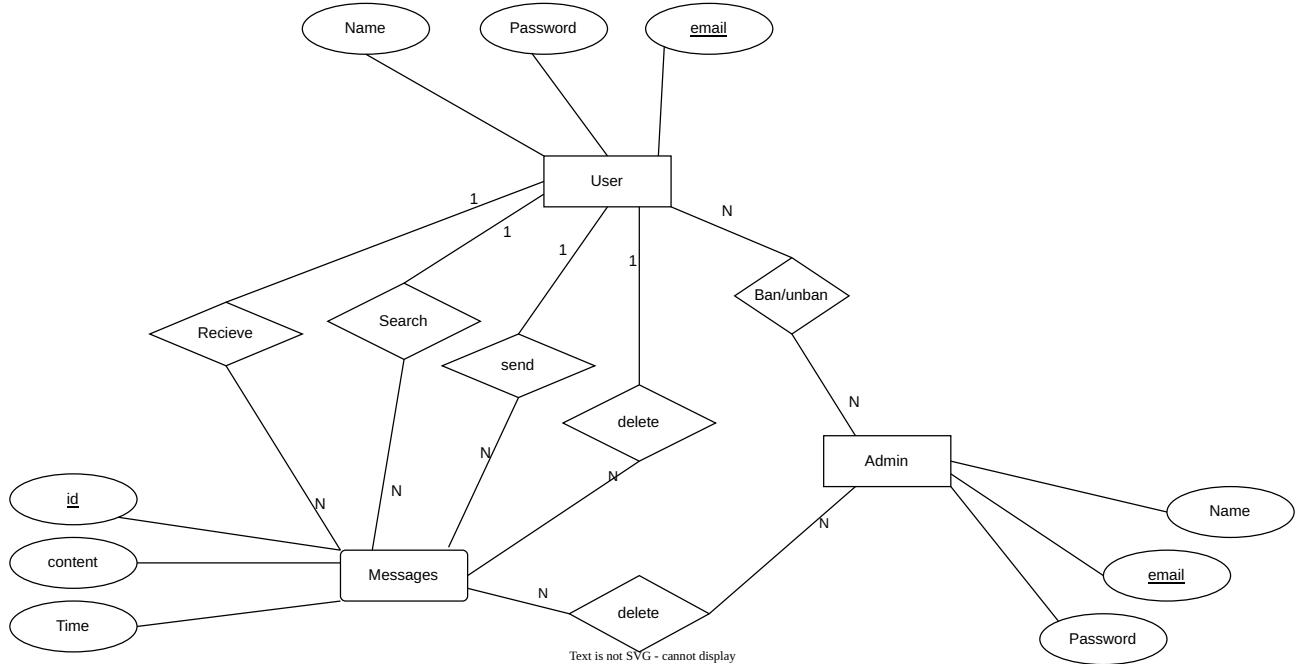


Figure 44: Entity Relationship Diagram

## 4.4 Application Screenshots

These are some screenshots to show the application interface and how it looks like.

### 4.4.1 Sign Up Page

This is the sign-up page where the user can create an account by providing his name, email, password, and profile picture.

The Sign Up page interface includes the following fields:

- Full Name
- Email
- Phone Number
- Avatar URL
- Password
- Confirm Password

Below the fields is a blue "Sign Up" button. At the bottom, it says "Already have an account? [Sign In](#)".

On the right side of the form, there is a decorative illustration of five stylized human figures in blue and white, each holding a smartphone, with speech bubbles above them.

Figure 45: Sign Up Page

### 4.4.2 Sign In Page

This is the sign-in page where the user can log in by providing his email and password.

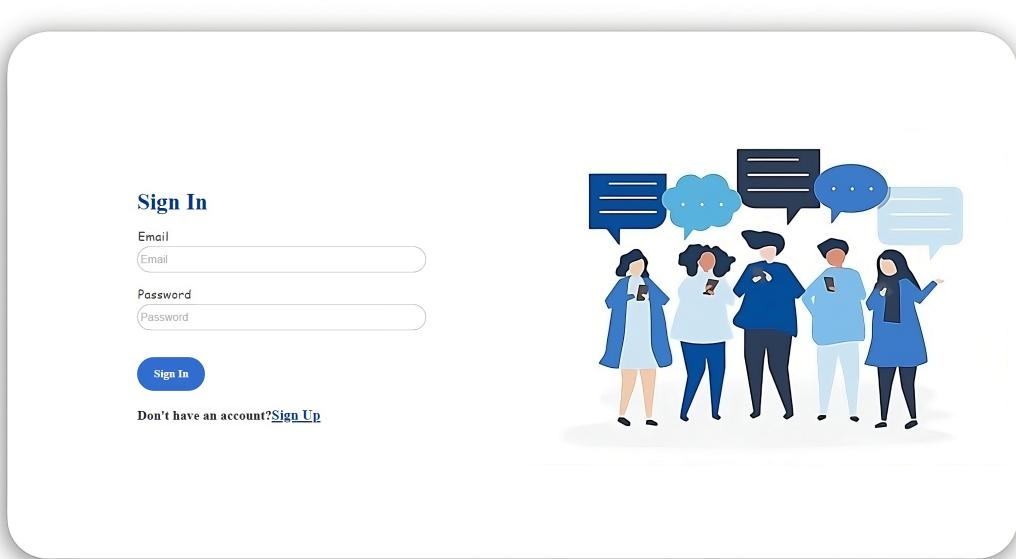


Figure 46: Sign In Page

### 4.4.3 Home Page

This is the home page where the user can start chatting with anybody.

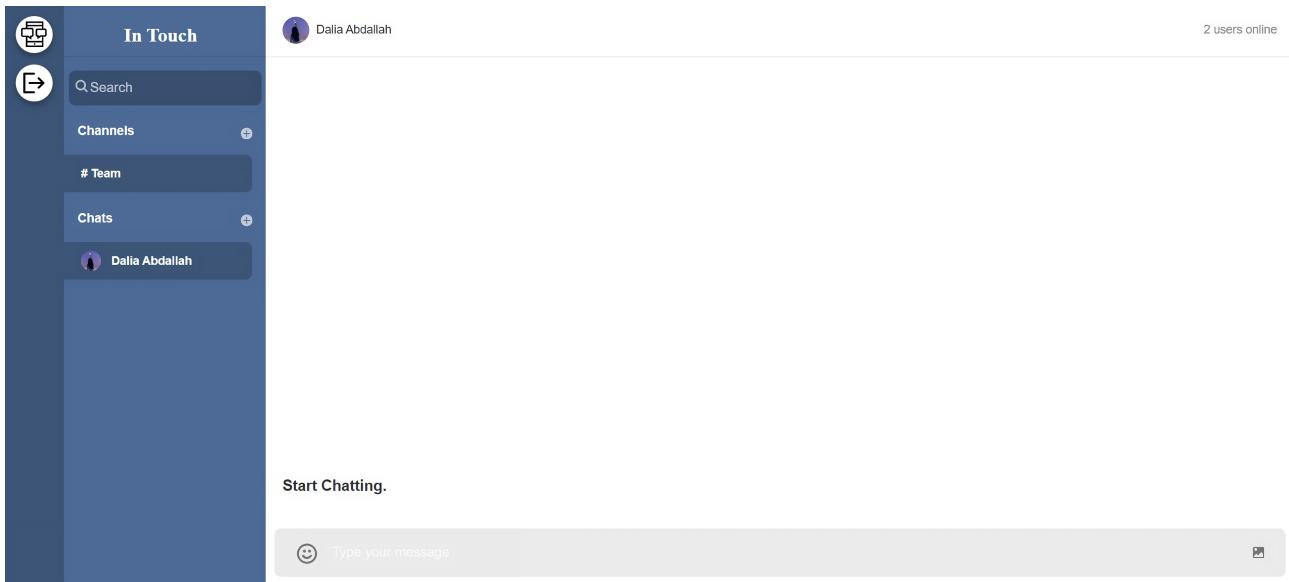


Figure 47: Home Page

### 4.4.4 Team Messages

This is the team messages page where the user can chat with multiple users at the same time.

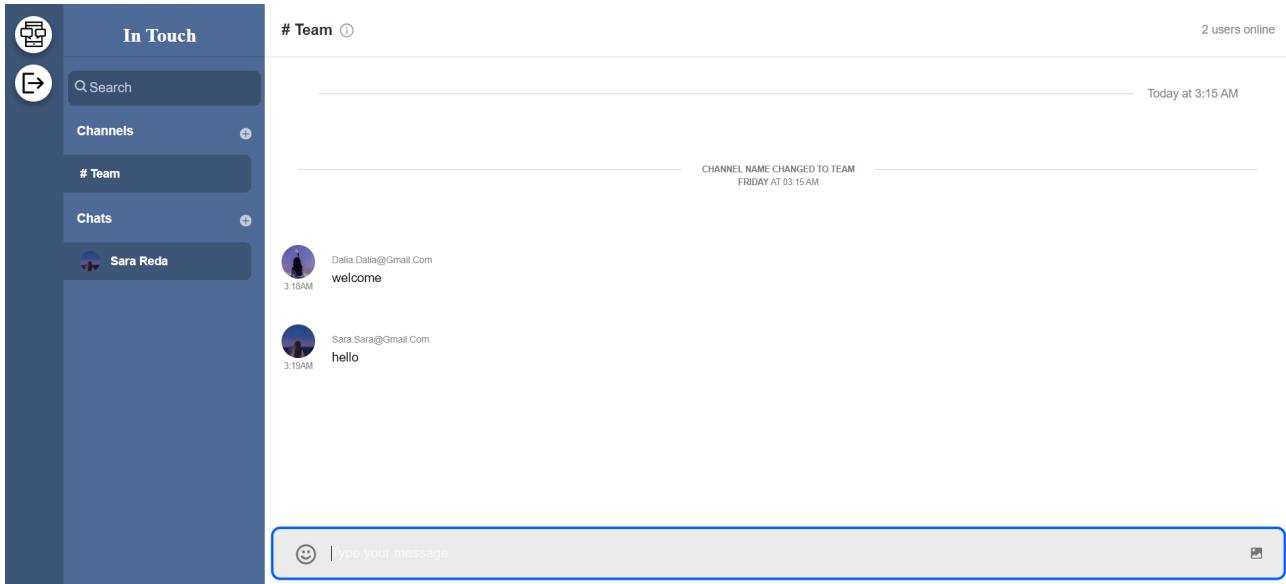


Figure 48: Team Messages

### 4.4.5 Messages in Chat

This is the normal look of the messages in chat.

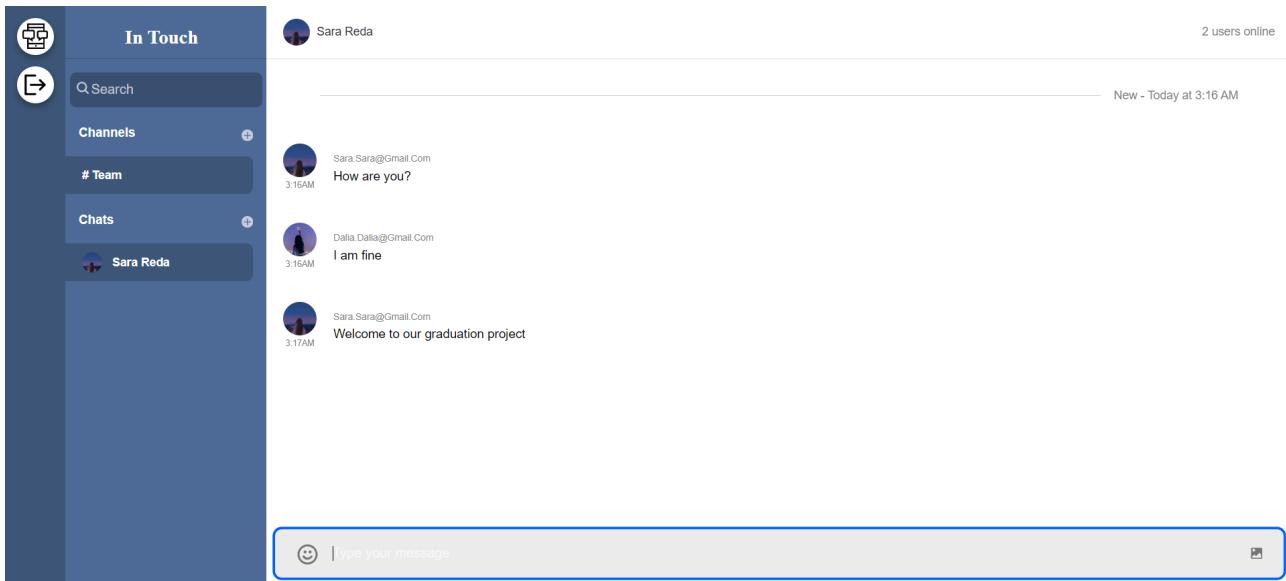


Figure 49: Chat

### 4.4.6 Messages Behind the Scenes

This is how the messages text looks like behind the scenes.

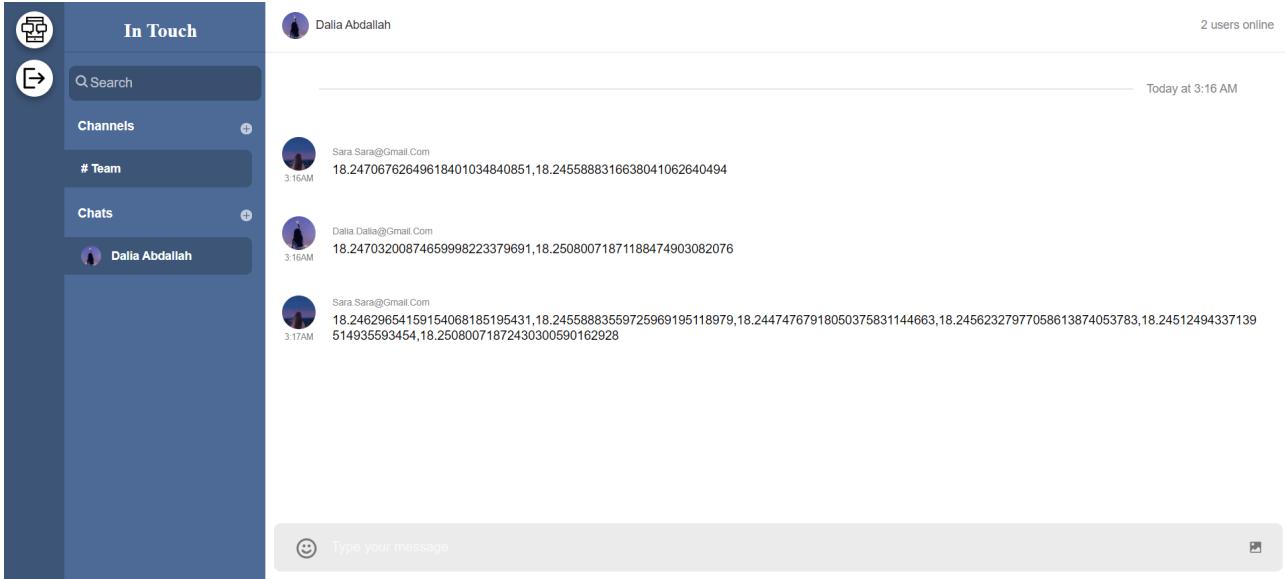


Figure 50: Messages Behind the Scenes

### 4.4.7 Messages in Stream

This is how the messages look like in the stream.js library.

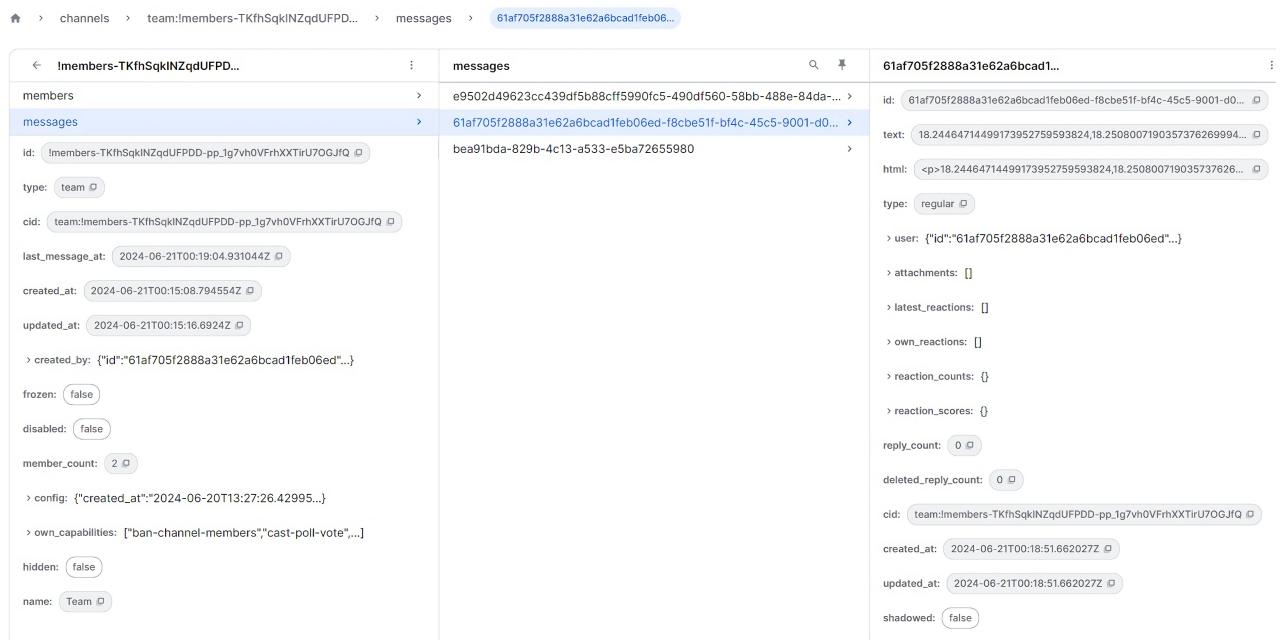


Figure 51: Messages in Stream

## 4.5 Tools & Libraries

### 4.5.1 Visual Studio Code

Visual Studio Code was used as the primary Integrated Development Environment (IDE) for the project. It's a source-code editor developed by Microsoft for Windows, Linux, and macOS. It includes support

for debugging, embedded Git control, syntax highlighting, intelligent code completion, snippets, and code refactoring.

### 4.5.2 Git

Git is a distributed version control system for tracking changes in source code during software development. It is designed for coordinating work among programmers, but it can be used to track changes in any set of files.

We used it to manage the project files and collaborate on the project.

**4.5.2.1  GitHub** GitHub is a web-based platform used for version control and collaboration. It provides features like bug tracking, feature requests, task management, and wikis for every project.

We used it to host the project repository and collaborate on the project.

### 4.5.3 Python

The algorithm was written in Python which is a high-level, general-purpose programming language. It is known for its simplicity and easy-to-read syntax. It is widely used in scientific and numeric computing, web development, and artificial intelligence.

The following libraries were used:

**4.5.3.1 `pandas`** Pandas is a software library for the Python programming language that provides data manipulation and analysis capabilities.

We used it to read and manipulate the datasets from the CSV files.

**4.5.3.2 `matplotlib`** Matplotlib is a plotting library for the Python programming language. It provides an object-oriented API for embedding plots into applications using general-purpose GUI toolkits like Tkinter, wxPython, Qt, or GTK.

We used it to create the plots for the results of the algorithm.

**4.5.3.3 `numpy`** NumPy is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays.

We used it to generate the points for most of the mathematical function and to generate the random numbers as it's much faster than the built-in python solutions.

**4.5.3.4 `scienceplots`** SciencePlots is a matplotlib style library that provides style sheets for plots to look like they would fit into a scientific publication.

It was used to make publication-ready plots.

**4.5.3.5 `scipy`** SciPy is a free and open-source Python library used for scientific computing and technical computing. It contains modules for optimization, linear algebra, integration, interpolation, special functions, FFT, signal and image processing, ODE solvers and other tasks common in science and engineering.

It has optimized built-in functions for the bisection and false position methods, and newton forward difference interpolation.

**4.5.3.6  time** The time module in Python provides various time-related functions. It is a part of Python's standard library and is used for handling time-related tasks like getting the current time, converting timestamps to readable formats, delaying the execution of functions, and more.

We used it to measure the execution time of each algorithm.

**4.5.3.7  string** The string module in Python includes functions to process standard Python strings. It contains constants for the printable ASCII characters, for various string operations, and for creating custom string transformations.

We used it to convert the plaintext message to an integer representation.

**4.5.3.8  random** The random module in Python is used to generate random numbers. It can be used to generate random integers, random sequences, and random choices. It can also be used to shuffle sequences.

**4.5.3.9  os** The os module in Python provides a way of using operating system-dependent functionality. The functions that the OS module provides allows you to interface with the underlying operating system that Python is running on – be that Windows, Mac or Linux.

We used it to get the current working directory and to read the CSV files.

**4.5.3.10  Decimal** The decimal module provides support for fast correctly-rounded decimal floating point arithmetic. It offers several advantages over the float datatype.

We used it to handle the decimal numbers in the algorithm.

### 4.5.4 Stream.js

Stream.js is a JavaScript library for building real-time web applications. It provides a simple and flexible API for building real-time messaging applications.

We used it to create the real-time messaging feature in the application.

### 4.5.5 HTML5

HTML5 is a markup language used for structuring and presenting content on the World Wide Web. It is the fifth and current major version of the HTML standard.

We used it to create the front-end of the application.

### 4.5.6 CSS3

CSS3 is the latest evolution of the Cascading Style Sheets language and aims at extending CSS2.1. It brings a lot of long-awaited novelties, like rounded corners, shadows, gradients, transitions, or animations, as well as new layouts like multi-columns, flexible box, or grid layouts.

We used it to style the front-end of the application.

### 4.5.7 JavaScript

JavaScript is a high-level, interpreted programming language that conforms to the ECMAScript specification. It is a lightweight, cross-platform, and open-source language.

We used it to create the interactive features of the application.

### 4.5.8 Node.js

Node.js is an open-source, cross-platform, back-end JavaScript runtime environment that runs on the V8 engine and executes JavaScript code outside a web browser.

We used it to run the server-side code of the application.

### 4.5.9 React

React is a JavaScript library for building user interfaces. It is maintained by Facebook and a community of individual developers and companies.

We used it to create the front-end of the application.

### 4.5.10 Markdown

Markdown is a lightweight markup language with plain-text-formatting syntax. Its design allows it to be converted to many output formats, but the original tool by the same name only supports HTML.

We used it to write the documentation.

### 4.5.11 Mermaid

Mermaid is a simple markdown-like script language for generating charts from text via JavaScript. It is used to create diagrams like flowcharts, sequence diagrams, and Gantt charts.

We used it to create the diagrams in this document.

### 4.5.12 Pandoc

Pandoc is a free and open-source document converter, widely used as a writing tool and as a basis for publishing workflows.

We used it to convert the markdown files to PDF via L<sup>A</sup>T<sub>E</sub>X.

## 5 References

1. jagpreet kaur, Dr. Ramkumar K.R.. A Cryptographic Algorithm using Polynomial Interpolations for Mitigating Key-Size Based Attacks, 14 September 2022, PREPRINT (Version 1) available at Research Square [<https://doi.org/10.21203/rs.3.rs-2050151/v1>]
2. Badr, El-Sayed & Attiya, Hala & El Ghamry, Abdallah. (2022). Novel hybrid algorithms for root determining using advantages of open methods and bracketing methods. Alexandria Engineering Journal. 61. 11579-11588. 10.1016/j.aej.2022.05.007.
3. Harder, D.W. Numerical Analysis for Engineering. Available online: <https://ece.uwaterloo.ca/~dwharder/nm/> (accessed on 11 June 2019).
4. Srivastava, R.B.; Srivastava, S. Comparison of numerical rate of convergence of bisection, Newton and secant methods. J. Chem. Biol. Phys. Sci. 2011, 2, 472–479.
5. Moazzam, G.; Chakraborty, A.; Bhuiyan, A. A robust method for solving transcendental equations. Int. J. Comput. Sci. Issues 2012, 9, 413–419.
6. Nayak, T.; Dash, T. Solution to quadratic equation using genetic algorithm. In Proceedings of the National Conference on AIRES-2012, Vishakhapatnam, India, 29–30 June 2012.
7. Calhoun, D. Available online: <https://www.boisestate.edu/math/>.
8. Ehiwario, J.C.; Aghamie, S.O. Comparative Study of Bisection, Newton-Raphson and Secant Methods of Root-Finding Problems. IOSR J. Eng. 2014, 4, 1–7
9. Mathews, J.H.; Fink, K.D. Numerical Methods Using Matlab, 4th ed.; Prentice-Hall Inc.: Upper Saddle River, NJ, USA, 2004; ISBN 0-13-065248-2.
10. Esfandiari, R.S. Numerical Methods for Engineers and Scientists Using MATLAB; CRC Press: Boca Raton, FL, USA, 2013.
11. Chapra, S.C.; Canale, R.P. Numerical Methods for Engineers, 7th ed.; McGraw-Hill: Boston, MA, USA, 2015.