A Comparative Study of the Blowfish and RSA Encryption Algorithms' speed while encrypting data of different sizes.

To what extent does the time complexity of RSA Encryption compare to Blowfish Encryption when encrypting data of varying sizes?

Subject: Computer Science

Word Count: 3950

Table Of Contents

1 Introduction

Cryptography is the study of secure communications techniques that allow the sender and intended recipient of a message to view its contents *(Kaspersky)*. In Computer Science, it refers to a field that utilizes mathematical equations and techniques to encrypt/decrypt data on the internet or stored within local computers *(Richards, Kathleen)*. This field has played an essential role in the development of Computer Science and our modern society since it has made transmission of information secure. The use of Cryptography is present everywhere in our daily lives in applications such as email clients, bank servers and social media applications. *(Ward, Mark)*. The output of research on Cryptography is high and large organizations such as Microsoft and IBM are major stakeholders in this research. *(Newsroom.ibm)*

Within the field of Cryptography, encryption algorithms practically apply cryptography's idea of securing data since they take in a plaintext file as an input and encrypt it into an encrypted file through complex algorithms *(Kath, Heath)*.

In this paper, I will focus on comparing the time complexity between 2 encryption algorithms which use different architectures: Blowfish and RSA while they encrypt data of varying sizes. Thus, the research question "To what extent does the time complexity of RSA Encryption compare to Blowfish Encryption when encrypting data of varying sizes?".

The research in this paper will be useful for advancing the speed of DNS (i.e Domain Name System) security on the internet. Nowadays, asymmetric key encryption is used to encrypt online data but since the algorithms it uses to encrypt data are computationally intensive *(Torres, Jessica)*, the speed of DNS security is slow. A slower deployment of DNS security means that attackers can seize control over incoming mail and webpages by forging DNS records *(Bernstein, D.J.)*, leading to a loss of data privacy. Through this research paper, utilizing symmetric key encryption to encrypt data could be considered as an alternative since they use algorithms which encrypt DNS records much faster. A faster speed of encryption is useful since DNS records can be encrypted faster, meaning that 3rd party attackers will have trouble launching attacks and thus, thousands of DNS attacks can be potentially prevented.

To investigate this comparative study, a graphical user interface was programmed (Appendix B) and made to encrypt varying sizes of data stored in my local computer using an industry standard encryption API called BouncyCastle *(Castle, Bouncy)* that provides implementations of Blowfish and RSA algorithms. The process of encrypting data was carried out 10 times and patterns in the amount of time taken to encrypt were analyzed. Mathematical graphs and Computer Science theory were used to explain the obtained results.

2 Background Research

2.1 Cryptography and its types

There are 3 main types of cryptographic algorithms: Asymmetric Key algorithms, Symmetric Key algorithms and Hash Functions. *(Mybestwriter)*. However, in the context of this paper, we will focus on distinguishing between asymmetric key algorithms and symmetric key algorithms.

Asymmetric key encryption utilizes 2 keys called Private and Public Keys and are created through generating large prime numbers and performing mathematical operations on them. *(Self-Defense, Surveilance)* Public Key as the name implies, is available to the public through a public directory such as the internet or a public key infrastructure server. Private Key on the other hand, is only available to the key's generator and is highly secure. If a public key encrypts the input message, the corresponding private key decrypts that encrypted message and vice - versa. This form of encryption is used in applications which secure and pass data through the internet from a sender to a receiver such as the SSL cryptographic protocols (ie. Secure Sockets Layer protocols which give secure connections) as well as Bitcoin. *(Brush, Kate)* The image below shows the mechanism of asymmetric key encryption:

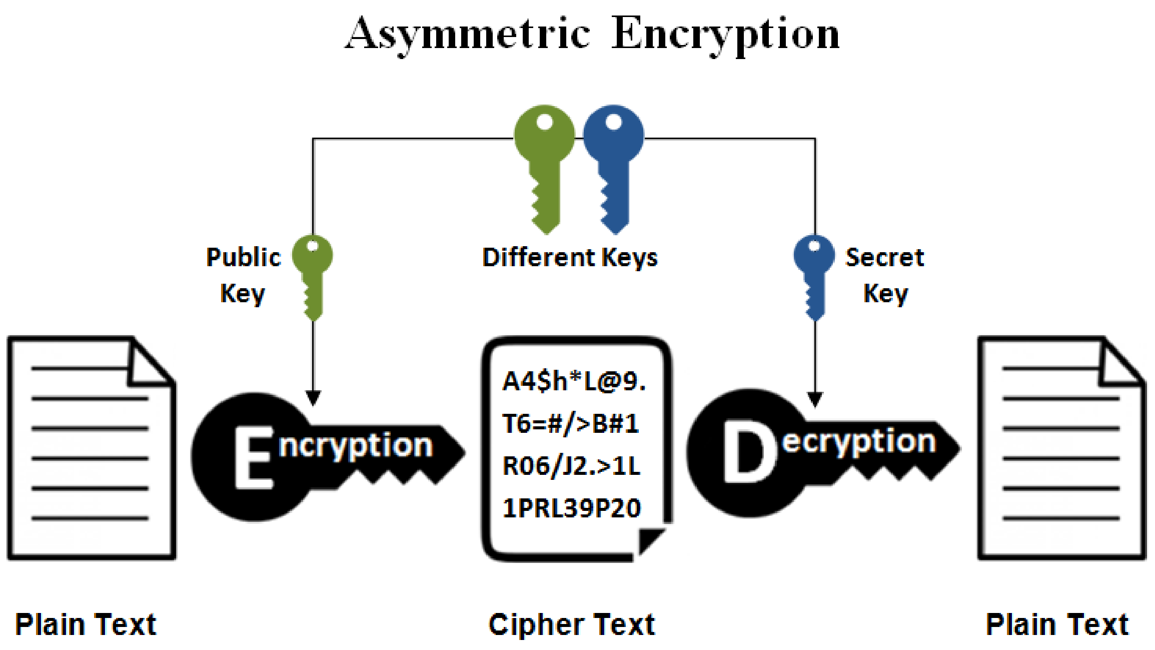


Figure 1: A diagram representing the encryption process of an asymmetric encryption algorithm (*Mishra, Neeraj*)

Symmetric Key encryption algorithms utilize the same key for encrypting and decrypting data. The key is commonly shared between 2 or more users *(Libfeld, Roey)*. The process of encryption runs the data through an encryption algorithm called a "cipher", which generates a ciphertext as output. 2 common symmetric encryption blueprints are Block and Stream Ciphers. Block Ciphers group data into static blocks and are encrypted using the key and algorithm of the same length as the block of data. Stream ciphers encrypt data by 1-bit increments. (1-bit plaintext data is encrypted into 1-bit ciphertext at a time) *(Academy, Binance)*. This form of encryption is used where the user encrypts large amounts of data in a short time to share with others. The image below shows the mechanism of symmetric key encryption:

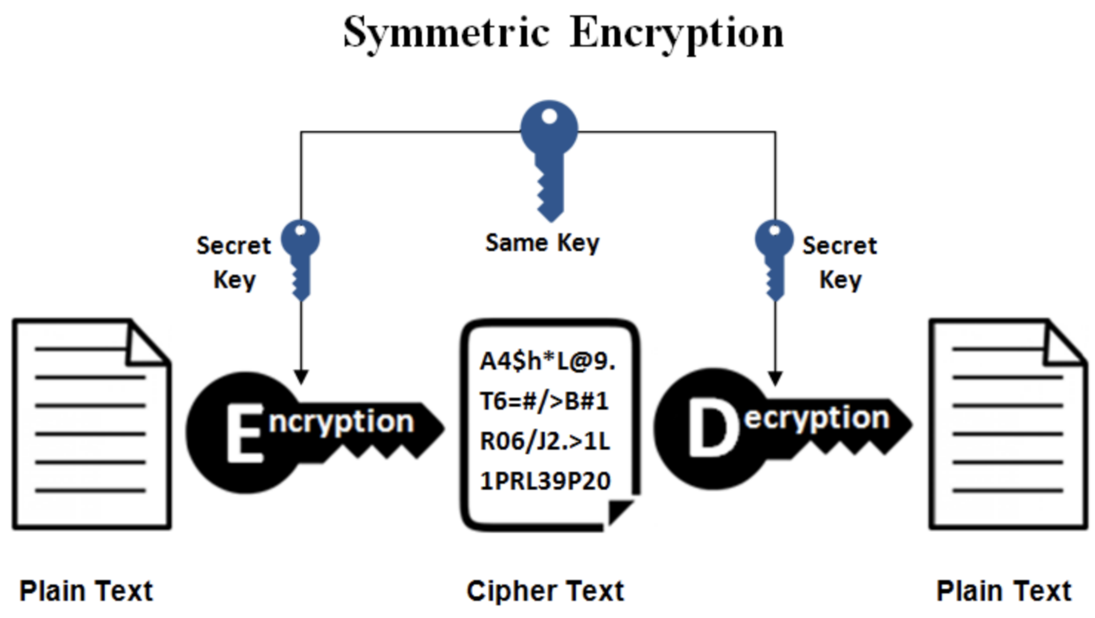


Figure 2: A diagram representing the encryption process of a symmetric encryption algorithm (*Mishra, Neeraj*)

2.2 RSA Encryption Algorithm Description

RSA encryption is short for Rivest Shamir Adleman encryption algorithm. The algorithm was created by Ron Rivest, Adi Shamir, and Leonard Adleman in 1977 *(Point, Tutorials)* and is used in data encryption of email and digital transactions through online retailers such as Amazon.

Since RSA is an asymmetric algorithm, both of its public and private keys are generated and linked together. These keys are generated through large prime numbers, the private key is the representation of 2 very large prime numbers whereas the public key is the product of the 2 prime numbers used to make the private key.

The encryption process of the RSA algorithm is shown below:

1. Two **random prime numbers are chosen**. The higher the digit of the prime number, the higher the security of the encrypted data.

   $Let\ a\ be\ the\ first\ prime\ number,\ Let\ b\ be\ the\ second\ prime\ number$

2. The next step is to then find out the **modulus**, using the following formula

   $modulus\ =\ a\ \cdot b$

3. Once we have the modulus, we need to use the **Carmichael's Totient Function** in order to find out the private key for the future.

   $lcm\ =\ lowest\ common\ multiple$
   $Let\ Dp\ be\ the\ product\ of\ the\ difference\ between\ the\ prime\ numbers\ from\ 1$
   $Dp\ =\ (a\ -\ 1)\ \cdot\ (b\ -\ 1)$
   $\lambda(modulus)\ =\ lcm\ \cdot\ (Dp)$

4. Now we have 1 piece of the public key, which is the **modulus**. We need to now get the other piece of the public key, which is commonly called e. We can assign e to any random value, the higher the value, the more efficient the encryption.

5. We now have both pieces of our public key, so we are ready to **encrypt** the plaintext/original data.

   $Let\ plaintext/original\ data\ be\ o$

   $Let\ encryptedText\ be\ c$

The formula to encrypt the original data uses modular exponentiation, and the formula for that is:

$c\ = o^{e}\ mod\ modulus$

6. Now that we have encrypted our data, it's time to **generate our private key** for decrypting the data. The private key can be generated by:

   $Let\ private\ key\ be\ d$

   $d\ =\ 1/e\ mod\ \ \lambda(modulus)$

***$1/e$ isn't dividing 1 by e, but calculating the modular multiplicative inverse of e**

7. We have now generated the private key, all that's left for us to do is to **decrypt the encrypted data.** This can be done by:

$$o = c^d \mod modulus$$

2.3 Blowfish Encryption Algorithm Description

Designed by Bruce Schneier in 1993, Blowfish is a symmetric block cipher algorithm, meaning that it rearranges data into static length blocks during the process of encryption and decryption *(ForGeeks, Geeks)*. Blowfish is used in file and disk encryption, password management, email encryption, etc *(Blowfish Encryption: Strength & Example).*

The encryption process of the Blowfish algorithm is shown below:

1. Initialize an array with 18 slots.

   $Let\ the\ array\ of\ 18\ slots\ be\ a$

2. Initialize 18 subkeys which will have 8 digits of hexadecimal, the hexadecimal values can be randomly generated but are constant values.

   $Let\ the\ subkeys\ 1 - 18\ be\ S_N$

3. Feed in all of the subkeys into the empty array of 18 slots.

   $a[N] = S_N$

4. Generate an input key which has a key length ranging from 8-56 digits of hexadecimal, the hexadecimal values are randomly generated but are constant values.

5. Perform the XOR operation on all the individual elements on the array to 32 bits of the input key. (It's important to note that if the input key is larger than 32 bits, then it's necessary to divide it by 32 bits to get the individual keys that are used to XOR the individual elements of the array instead of the original input key. If there are no more individual keys of length 32 bits remaining, then XOR the remaining elements of the array from the beginning of the input keys until the array has finished being initialized) *(Sharma, Abhishek)*.

   $a[N] = a[N] \oplus 32\ bits\ of\ input\ key$

6. Now that the array of 18 slots has been initialized, an array of 4 substitution boxes is initialized, ($S[0]$, $S[1]$, $S[2]$, $S[3]$). Each of the elements in the array will have 256 entries that will be 8 digits of hexadecimal each. (In total, each of the entries in the substitution box will have 2048 digits of hexadecimal). The hexadecimal values are randomly generated but are constant values.
7. Now, we can start encrypting the input data by splitting it into 64 bit blocks and then passing it through a **Feistel Network Structure.**

The diagram for the Feistel Network Structure that the Blowfish Encryption Algorithm uses is shown below:

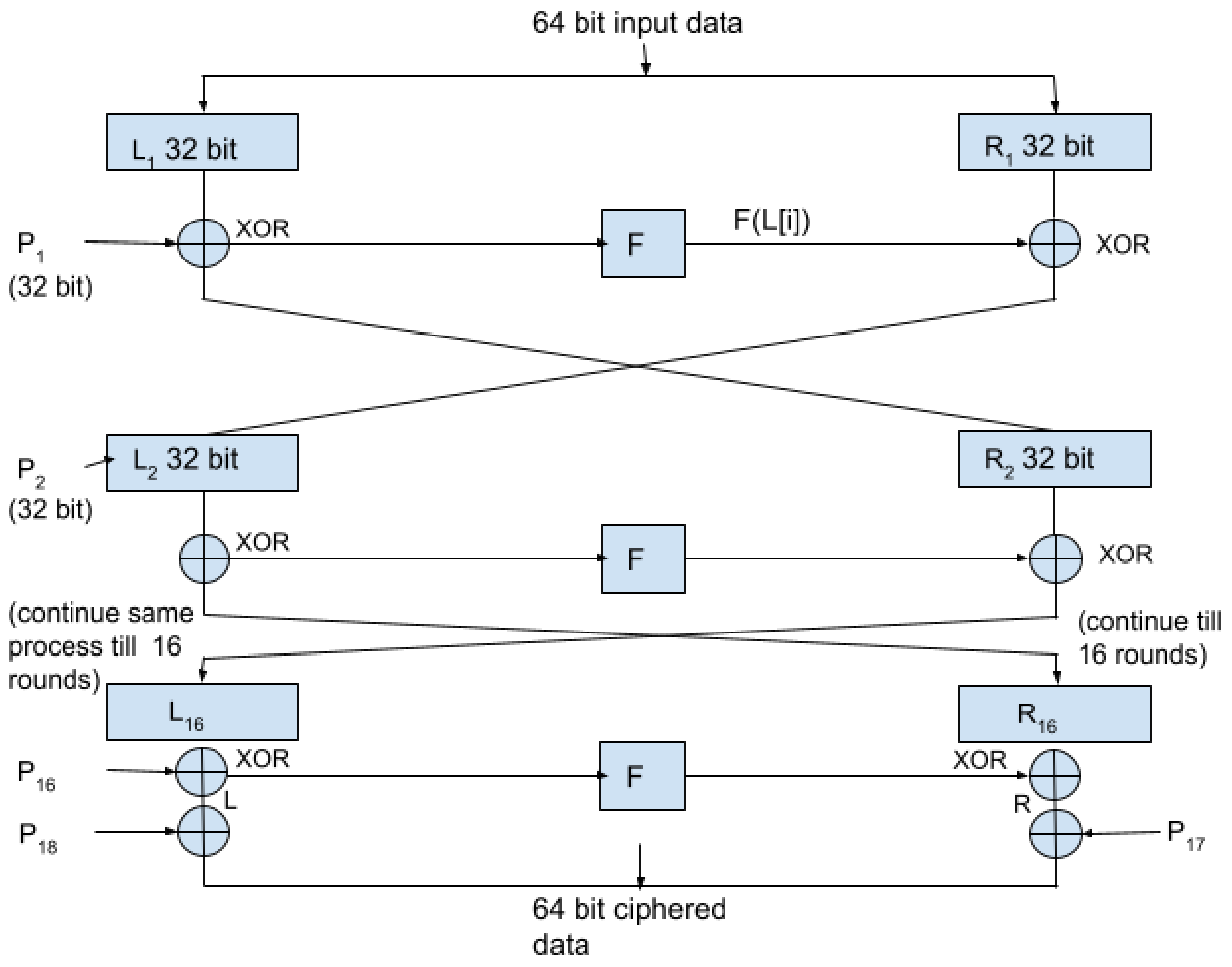


Figure 3: Diagram showing the Inner Workings of the Feistel Network Structure

A **Feistel Network Structure** is a blueprint from which many different block ciphers are derived. Any block cipher algorithm that utilizes a Feistel Structure will have the same process for encrypting and decrypting a plaintext, the steps that it takes to encrypt the input text of Blowfish is shown below:

1. It will first split the 64 input data block into two 32 bit blocks which are fed into 2 halves, $(L \ and \ R)$

2. This data will then be propagated through the first round in the network which is then performed an XOR operation with the first element of the array a and then passed into the function **F**, producing the output of **F(L[0])** and then performed an XOR mathematical operation with the other 32 bits of data located on the R side of the network. When the operations on the first round are finished, the final values produced in the L and R sides of the network are then swapped and propagated into the next round.
3. The process in step 2 repeats until the end of the 16th round when the values on the L and R sides of the network are swapped again and then finally concatenated to get the ciphered output text.

The diagram of the **Function F**, that the Feistel Network Structure uses is shown below:

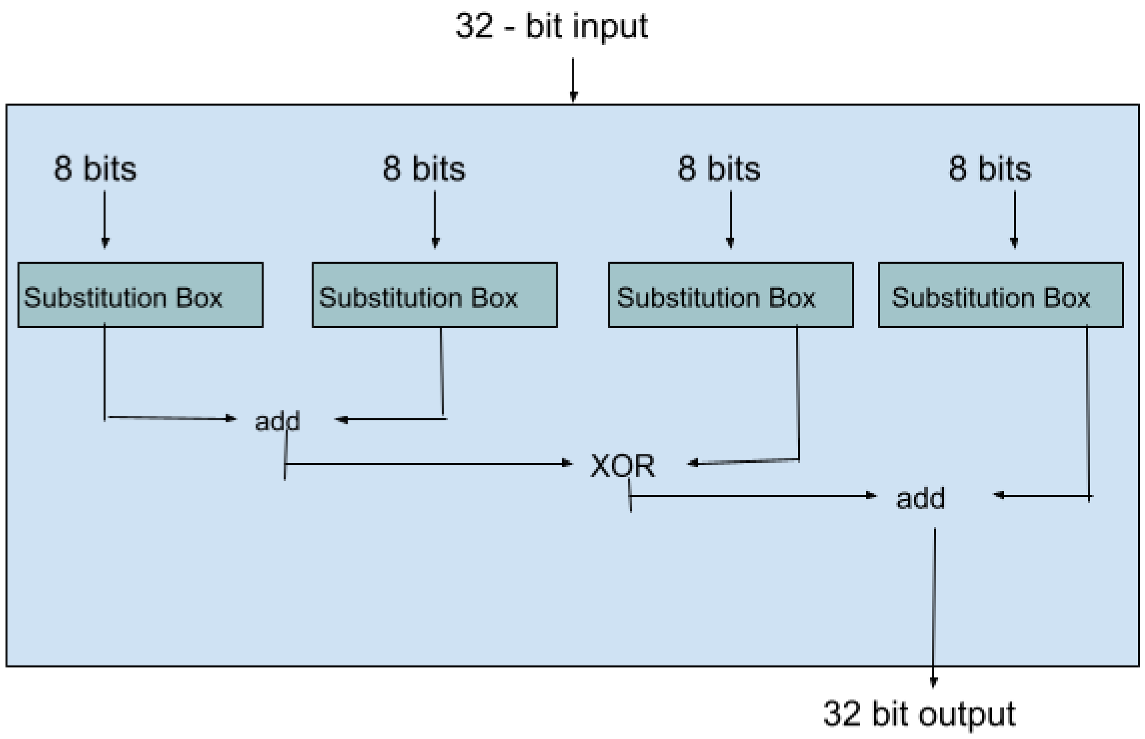


Figure 4: Diagram showing the inner workings of the Function F used in the Feistel Network Structure

The Function **F** in the Feistel Network Structure works by:

1. Taking in a 32-bit input and splitting it into four 8-bit blocks.
2. Feeding in the four 8-bit blocks into substitution boxes (Substitution boxes makes the relationship between the key and cipher text non linear, and plays a part in securely encrypting the data) *(Chandrasekaran, Jeyamala, et al.)*.
3. Adding the values of the first 2 substitution boxes, then performing the XOR mathematical operation with the value on the 3rd substitution box. Then adding the value from the 3rd substitution box value to the 4th substitution box to finally get a 32-bit output.

2.4 Time Complexity and Big O Notation of the Encryption Algorithms

Time complexity is defined as the amount of time taken by an algorithm to run corresponding to the length of the input *(Cormen, Thomas)*. It gives us an indication of the execution time of an algorithm depending on the input size; the length of the input data highlights the number of operations to be performed by the algorithm *(GreatLearning)*. One of the most widely used ways to measure time complexity is through the Big O notation. The Big O notation consists of 2 parts Θ and n , where Θ is the order of growth and n is the length of the input. Therefore, time complexity denoted by the Big - O notation can be written as $\Theta(n)$. It should be noted that RSA has a time complexity of $\Theta(n^2)$ *(Hub, Algo)* and Blowfish has a time complexity of $\Theta(n)$ *(Saranya, V., and K. Kavitha)*. The general information about the time complexities of these algorithms are explained below:

$\Theta(n)$- This represents that the order of growth corresponding to the input is in **linear time**, and that running time increases similarly with the length of the input.

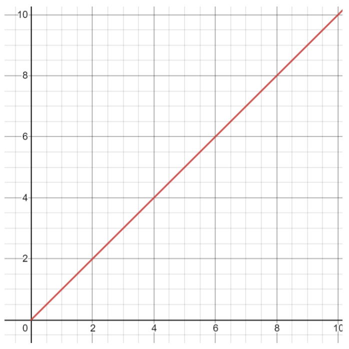


Figure 5: Graph showing the $\Theta(n)$ time complexity

$\Theta(n^2)$ - This represents that the order of growth corresponding to the input is in **quadratic time**, and that running time increases non linearly n^2 with the input length.

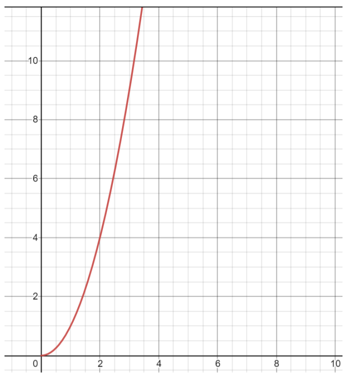


Figure 6: Graph showing the $\Theta(n^2)$ time complexity

3 Experimental Methodology

The methodology that will be used in this paper is the experimental method. The main reason why this methodology was chosen is because of its ability to manipulate the independent variable which is especially useful in this paper since I want to identify the cause of why encryption algorithms take more time as the sizes of different types of data increases. However, a limitation of this methodology is that data can be unequally distributed which is poor experimental design since it leads to increased outliers in data. The primary experimental data is the main source of data in this paper, an RSA and Blowfish algorithm was programmed (code in Appendix A, using the Bouncy Castle Library (*Castle, Bouncy*)) and fed in text files generated from a public dataset. A GUI was created to interact with the algorithms and view the results (refer to Appendix B for design).

3.1 Datasets Used and Processed

The datasets used in this paper were plaintext files that all had varying amounts of text generated from a public website called Lorem Ipsum *(Ipsum, Lorem)*. In Section 3.2, the amount of varying bytes that would be required in the experiment were detailled, and those amount of bytes were used to generate 10 random texts which were then exported as text files and ten copies were made which corresponded to the amount of trials in the experiment. These files were then fed into the algorithms, and the results were processed and analysed.

3.2 Independent Variable

The independent variable being measured is the **size of the data**. Each set of data are text files whose sizes are incremented by 10 kilobytes each, starting from 10 kilobytes and ending at 100 kilobytes. 10 sets of data will provide enough data points to be plotted on a 2D graph and observe the encryption algorithms' behaviour but will not be overwhelmingly large so that the experimental procedure becomes convoluted.

3.3 Dependent Variable

The dependent variable being measured in this experiment is the **amount of time it takes for the encryption algorithms to encrypt data of varying sizes**. This will be measured by utilizing the Stopwatch class and after encryption, the time will be displayed in the program in milliseconds. Displaying time in milliseconds is the most precise form of measurement that is available in the Stopwatch class.

3.4 Controlled Variables

Refer to Appendix D for the controlled variables in the experiment.

3.5 The Experimental Procedure

The procedure for the experiment is as follows:

1. Open Visual Studio 2019 and start the program, insert text files into the program periodically based on their sizes (10-100 KB) and then encrypt each of the files back and forth using the Blowfish and RSA encryption methods.
2. Repeat step 1 for all of the 10 trials until all the times of the text files of varying sizes have been recorded with each encryption algorithm.
3. Calculate the mean times for all of the encrypted text files with each encryption algorithm and then insert those values into a processed table which will be presented in the data collection stage of the paper.

4 Hypothesis and Theory

Now that we have covered the experiment's details and explained the background of each algorithm, the next step is to determine which encryption algorithm is more efficient while encrypting the data. This can be determined by comparing the time complexity of the algorithms (refer to section 2.4). As mentioned in Appendix D, this experiment will take into account that the data being passed through the RSA algorithm will be processed in blocks of 200 bits, whereas the source stating that RSA has a time complexity of $\Theta(n^2)$doesn't take this factor into account. As a result, the time complexity of RSA in this experiment may differ.

It was mentioned in section 2.3 that Blowfish uses a Feistel Network Structure which includes the function F. Both these structures perform bitwise operations to encrypt data whereas in section 2.2 it was mentioned that RSA uses modular arithmetic to encrypt data. Bitwise operations are less costlier than Modular Arithmetic since they perform less CPU Operations. Summing up all of these factors, it can be predicted that encrypting data with the Blowfish algorithm will take much less time to encrypt data compared to RSA.

This experiment will measure the relationship between time, y, and size of sets being inserted, x. By varying the size of the sets of values inserted into the encryption algorithms, a relationship between these variables can be determined and how this relationship differs between Blowfish and RSA should be observed.

Taking into consideration of all of the factors above, I hypothesize that the Blowfish encryption algorithm will have a time complexity of $\Theta(n)$ and the RSA encryption algorithm will have a time complexity of $\Theta(n^2)$. I also believe that Blowfish will take much shorter times to encrypt data compared to RSA.

5 Data Processing and Graphs

5.1 Data Collection and Processing

Below shows the mean times for all of the data sets that have been tested. For the raw data that displays times for all 10 trials, please refer to Appendix C.

	Average Time (milliseconds)	
File Size (KB)	**Blowfish Algorithm**	**RSA Algorithm**
10	0.21064	130.68711
20	0.41839	224.97667
30	0.62572	353.23467
40	0.80175	596.46189
50	1.08937	817.11564
60	1.27587	1064.47159
70	1.40936	1207.97618
80	1.66577	1476.38887
90	1.84731	1632.91879
100	1.99947	1822.91281

Figure 7: Average Encryption Times of Files tested for both encryption algorithms

5.2 Blowfish's Graph of Time vs File Size

The graph below shows the relationship of time against sizes of the files fed into the Blowfish algorithm.

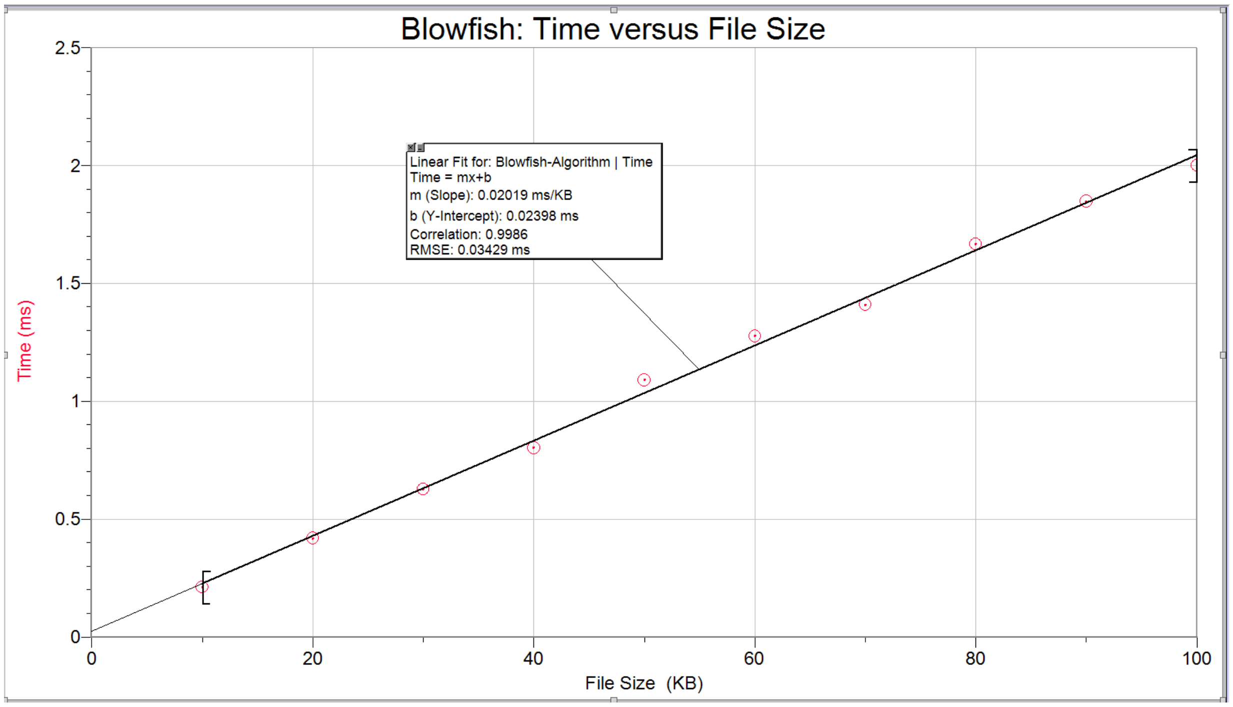


Figure 8: Graph showing Blowfish's graph of time vs file size.

5.3 RSA's Graph of Time vs File Size

The graph below shows the relationship of time against sizes of the files fed into the RSA algorithm.

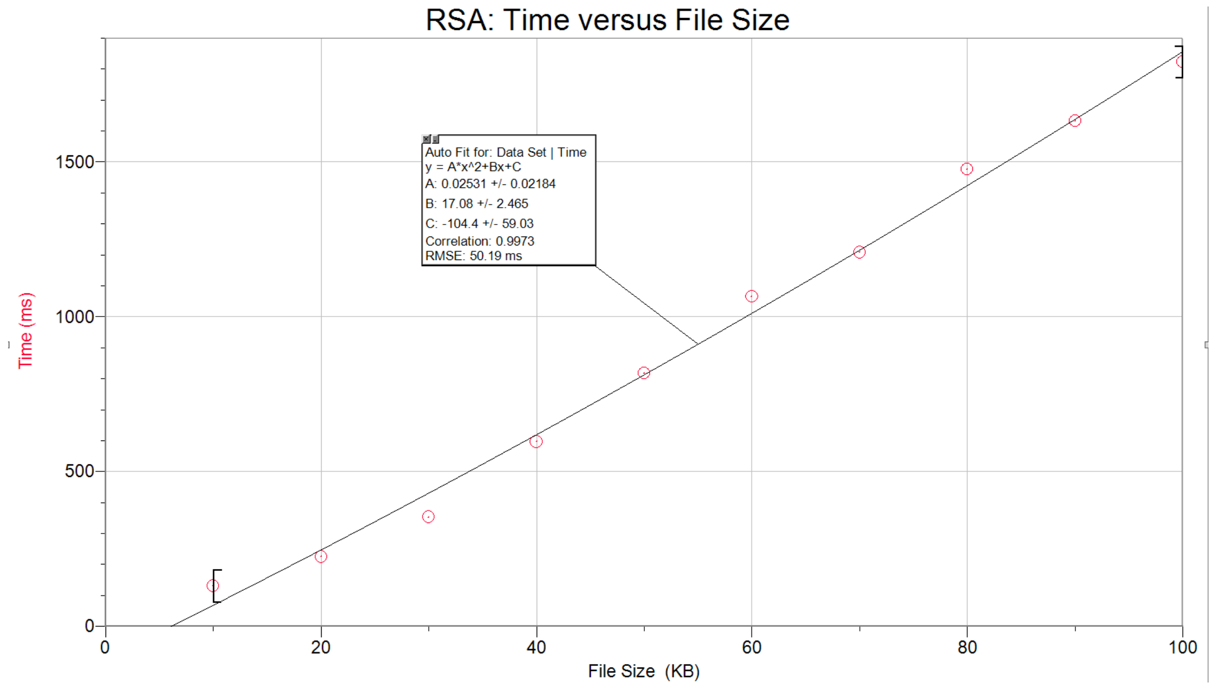


Figure 9: Graph showing RSA's graph of time vs file size.

5.4 RSA Vs Blowfish Bar Chart

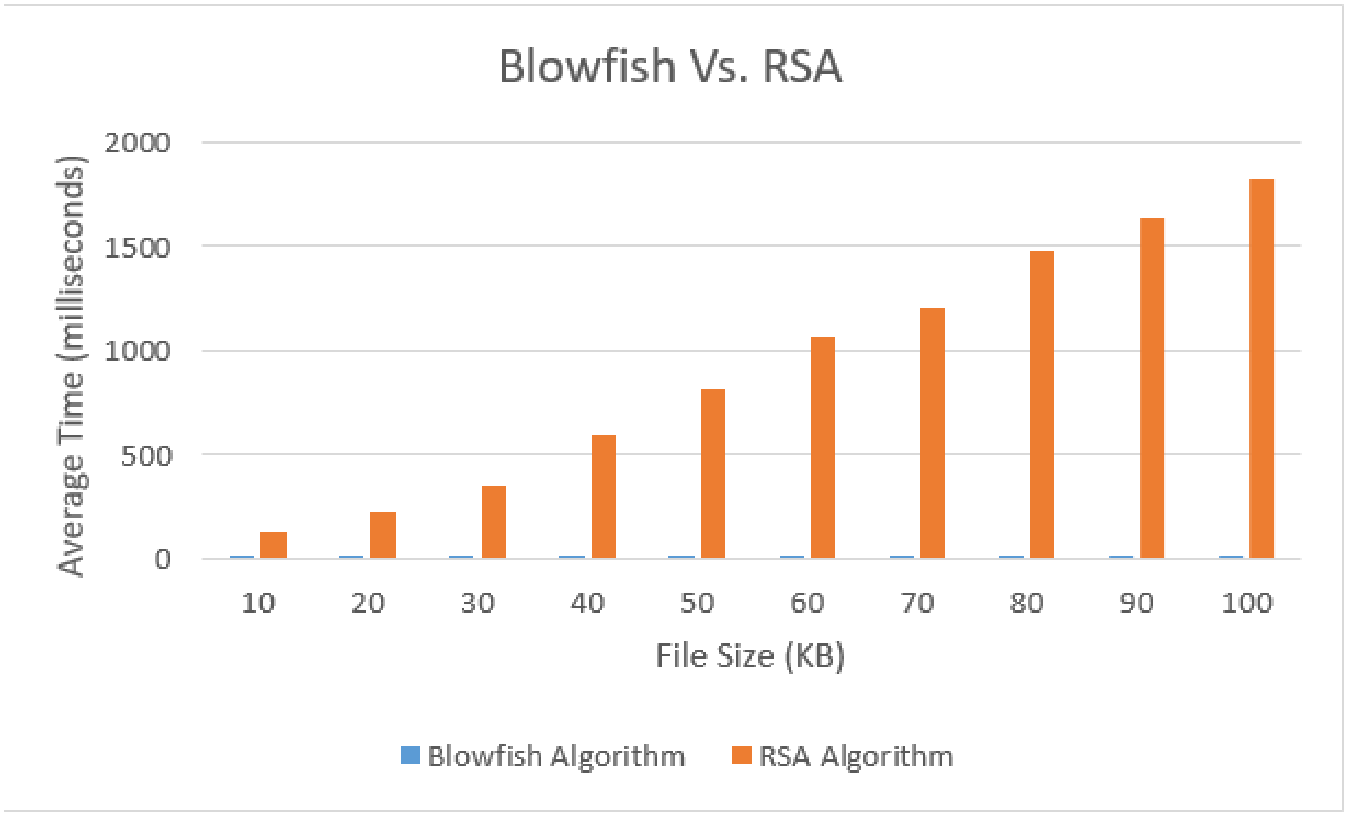


Figure 10: Bar Chart Comparison between Blowfish and RSA's time vs file size

6 Evaluation of the Data

My hypothesis which predicted that Blowfish would have a time complexity of $\Theta(n)$ and RSA would have a time complexity of $\Theta(n^2)$ have been proven as seen from Figure 8 and 9 respectively since Figure 8 has a linear fit that has a correlation of 0.9986 and the root square mean error of 0.03429 milliseconds suggests that the data points are highly concentrated around the line of best fit which supports my argument that Blowfish is an algorithm which running time increases linearly n to the length of the input. Figure 9 has a quadratic fit that has a correlation of

0.9973 and the root square mean error of 50.19 milliseconds indicates that the data points are highly concentrated around the line of best fit, supporting the argument that the algorithm's running time increases non linearly n^2 with the input length.

My other hypothesis which stated that Blowfish will take much shorter times to encrypt data has also been proven as seen from Figure 10. However, when I looked at Figure 10, I observed that there was a huge disparity between the encryption times of the 2 algorithms on all sets of data. As an example, for the 40KB file, Blowfish took 595.66014 milliseconds less than RSA! To find out the exact multiplier Blowfish was faster than RSA for all sets of data, I divided the time taken for the RSA from Blowfish to encrypt all sets of data and found the mean. I was shocked to see that **Blowfish was faster than RSA by a multiplier of 758.396!**.

I initially linked this result to step 2 of section 3.5 which stated how I will collect data for the experiment. The times recorded for encryption were done manually, as a result **random errors** could have occurred while recording the times and thus, the chances of outliers in the data were increased which affected the average encryption times.

I then linked this result to the background research I did on Blowfish on section 2.4 and wondered if this is due to the fact that Blowfish encrypts data through a Feistel Round Structure (Figure 3) that uses the function F (Figure 4) both of these sub-modules of the algorithm utilize the bitwise operations of ADD and XOR. It should be noted that both of these operations are extremely fast and have the bit complexity of $\Theta(1)$ *(What Is the Complexity of Bitwise*

Operations?). RSA on the other hand mainly uses Modular Exponentiation and Multiplication while encrypting data (refer to the background research on section 2.2). It should be noted that these operations take the bit complexity of $\Theta((log\ n)^3)$and $\Theta((log\ n)^2)$respectively *(Menezes, A J, Oorschot P. C. Van, and Scott A. Vanstone)*. A graph comparing the bit complexities of these operations are shown below:

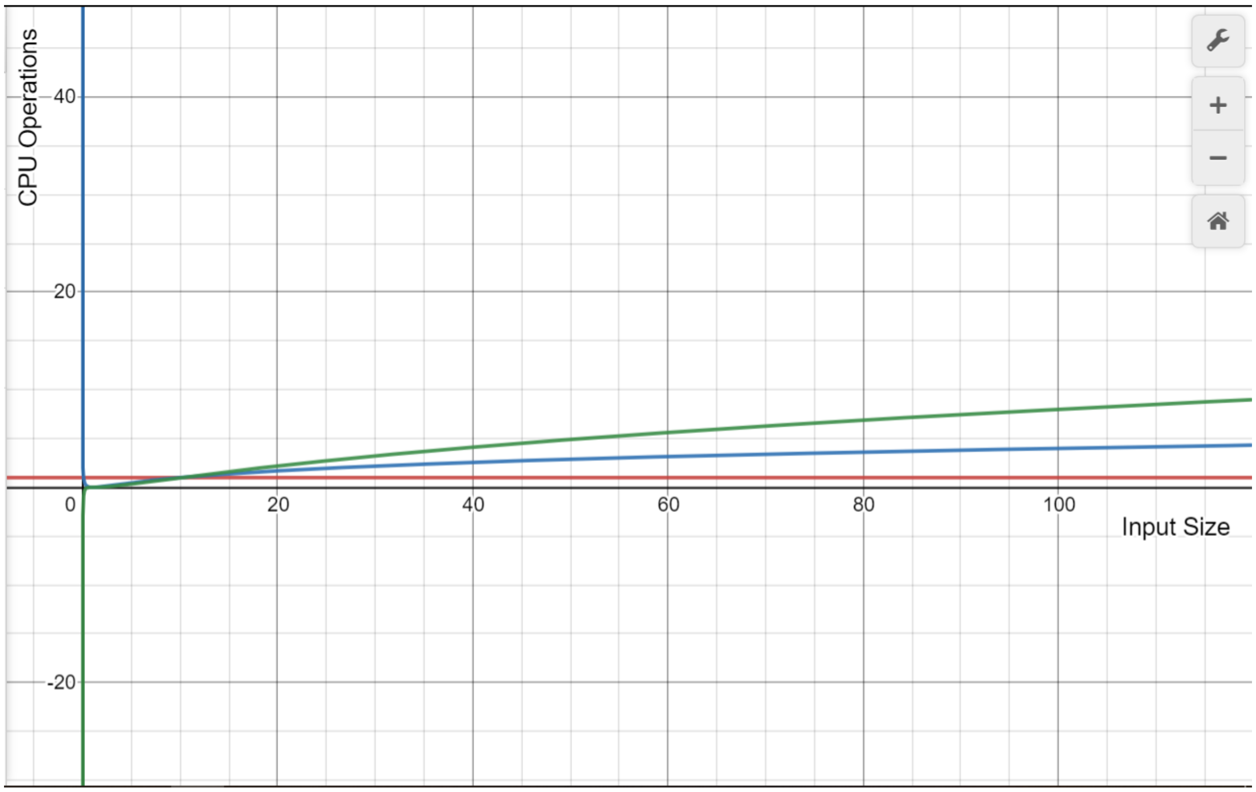


Figure 11: Graphs of $\Theta(1)$, $\Theta((log\ n)^2)$, and $\Theta((log\ n)^3)$

$\Theta(1)$ [XOR and ADD Operations] = Red Graph

$\Theta((log\ n)^2)$[Modular Multiplication Operation] = Blue Graph

$\Theta((log\ n)^3)$[Modular Exponentiation Operation] = Green Graph

As seen from Figure 11, it can be seen that modular multiplication and exponentiation start outgrowing the bitwise operations in terms of CPU Operations performed with increasing input size. This means that the CPU operations of $\Theta((log\ n)^2)$ and $\Theta((log\ n)^3)$ will take far more CPU Operations than $\Theta(1)$, which only performs 1 CPU Operation at any given input size and as a result, it can be said that RSA produces much larger encryption times than Blowfish in the long run.

7 Conclusion

This paper aimed to use background information of Blowfish and RSA encryption algorithms and apply it to determine the relationship between encryption times and sizes of the text files inserted into the encryption algorithms. As expected, there is a linear relationship between the time and sizes of the text files inserted for the Blowfish algorithm which is apparent in Figure 8 and it's apparent that there is a quadratic relationship between the time and sizes of the text files for the RSA algorithm which is visible in Figure 9. To further the scope of the paper, the investigation utilized the theory behind the encryption algorithms to justify the results of their time - file size relationship.

The data was incremented in sizes of 10KB from 10KB to 100KB to ensure that both the encryption algorithms were given plenty of varying sizes of data to encrypt and patterns from the amount of time both algorithms took could be analysed. As the above results and explanation show, due to the higher amounts of CPU Operations that the RSA's modular exponentiation and multiplication use compared to Blowfish's XOR and ADD operation, **I am concluding that Blowfish is a faster and more efficient encryption algorithm compared to the RSA encryption algorithm for all sets of data by an extremely large multiplier of 758.396.**

Hopefully, the research in this paper will be useful for advancing the speed of DNS Security by encouraging software developers that constantly encrypt data through asymmetric encryption algorithms to consider utilizing symmetric encryption algorithms like Blowfish, DES, Twofish. Additionally, through this research, developers could also consider encrypting data with symmetric encryption algorithms and then encrypting the key of that symmetric encryption algorithm with an asymmetric encryption algorithm (i.e hybrid encryption). That being said, the use of asymmetric encryption in cryptography is still hard to replace with symmetric cryptography due to the mechanism of asymmetric encryption encrypting data much more securely and giving less room for 3rd parties to crack and intercept.

8 Future Research

An important criteria for assessing the performance of an encryption algorithm is to test how securely it encrypts the data. *(Stine, Kevin, and Quynh Danh)*. Comparing RSA and Blowfish encryption algorithms in terms of their strength could be a suitable further research since it provides another benchmark to assess the algorithms in terms of efficiency.

To measure the strength of an encryption algorithm, the key sizes and the architecture it uses while encrypting data needs to be analyzed. In this paper, RSA had a larger key length and encrypted data in larger block sizes of 200 bits. The mechanism that RSA used to encrypt data performed mathematical operations on prime numbers, which produced cipher texts that were harder to crack compared to Blowfish. Thus, it can be concluded that although the Blowfish encryption algorithm took less time to encrypt data, RSA outperformed Blowfish by producing cipher texts which were more secure.

9 Works Cited

Academy, Binance. "What Is Symmetric Key Cryptography?" *AcademyBinance*, 8 Apr. 2019, academy.binance.com/en/articles/what-is-symmetric-key-cryptography. Accessed 8 Apr. 2019.

Bernstein, D.J. "High-speed Cryptography." *Cr.yp.to*, 9 Feb. 2006, cr.yp.to/highspeed.html. Accessed 15 Sept. 2021.

"Blowfish Encryption: Strength & Example." Study.com, 1 June 2016, study.com/academy/lesson/blowfish-encryption-strength-example.html

Brush, Kate. "Asymmetric Cryptography (Public Key Cryptography)." *SearchSecurity*, 10 May 2011, searchsecurity.techtarget.com/definition/asymmetric-cryptography. Accessed 25 July 2021.

Castle, Bouncy. *Bouncy Castle C# API*. *Bouncycastle.org*, 7 Dec. 2003. Accessed 18 June 2021.

Cormen, Thomas, et al. "Untitled post." *KhanAcademy*, Khan Academy, www.khanacademy.org/computing/computer-science/algorithms/asymptotic-notation/a/big-o-notation. Accessed 23 July 2021.

ForGeeks, Geeks. "Blowfish Algorithm with Examples." *GeeksforGeeks*, 14 Oct. 2019, www.geeksforgeeks.org/blowfish-algorithm-with-examples/.

GreatLearning, Great Learning, 24 Mar. 2020, www.mygreatlearning.com/blog/why-is-time-complexity-essential/.

Hub, Algo. "Rsa Cryptography Algorithm." *AlgorithmHub - RSA Cryptography Algorithm*, 2017, web.archive.org/web/20190614084338/algohub.me/algo/rsa-cryptography-algorithm.html.

Ipsum, Lorem. “Lorem Ipsum.” *Lorem Ipsum - All the Facts - Lipsum Generator*, 3 Dec. 2001, www.lipsum.com/.

Kaspersky. “Cryptography Definition.” *Www.kaspersky.com*, 13 Jan. 2021, www.kaspersky.com/resource-center/definitions/what-is-cryptography.

Kath, Heath. "How Encryption Works: Everything You Need to Know." *GoAnywhere*, HelpSystems, 28 July 2021, www.goanywhere.com/blog/how-encryption-works-everything-you-need-to-know. Accessed 15 Sept. 2021.

Libfeld, Roey. "What Is Symmetric Key Cryptography Encryption?: Security Wiki." *Secret Double Octopus*, 9 Feb. 2019, doubleoctopus.com/security-wiki/encryption-and-cryptography/symmetric-key-cryptography/.

Mishra, Neeraj. "Types of Cryptography." *TheCrazyProgrammer*, 19 Sept. 2020, www.thecrazyprogrammer.com/2019/07/types-of-cryptography.html. Accessed 26 July 2021.

Mybestwriter. MyBestWriter, 14 Sept. 2019, mybestwriter.com/three-main-types-of-cryptographic-algorithms/. Accessed 6 Aug. 2021.

Menezes, A J, Oorschot P. C. Van, and Scott A. Vanstone. Handbook of Applied Cryptography. Boca Raton: CRC Press, 1997.

Newsroom.ibm. 15 Mar. 2021, newsroom.ibm.com/IBM-Explores-the-Future-of-Cryptography. Accessed 14 Sept. 2021.

Point, Tutorials. "Understanding RSA Algorithm." *Understanding Rsa Algorithm*, 26 Nov. 2018, www.tutorialspoint.com/cryptography_with_python/cryptography_with_python_understanding_rsa_algorithm.htm.

Richards, Kathleen. "Cryptography." *SearchSecurity*, 19 Mar. 2011, searchsecurity.techtarget.com/definition/cryptography.

Saranya, V., and K. Kavitha. *A Modified Blowfish Algorithm for Improving the Cloud Security*. 22 Apr. 2018. *Ejerm*, www.ejerm.com/vol4_june_2017/img/pdf/3.A-Modified-Blowfish-Algorithm-for-Improving-the-Cloud-Security.pdf. Accessed 14 Sept. 2021.

Self-Defense, Surveilance. "A Deep Dive on End-to-End Encryption: How Do Public Key Encryption Systems Work?" *Surveillance Self-Defense*, 10 Aug. 2021, ssd.eff.org/en/module/deep-dive-end-end-encryption-how-do-public-key-encryption-systems-work#:~:text=The%20public%20key%20and%20private,very%20large%20secret%20prime%20numbers.

Sharma, Abhishek, narrator. *Blowfish Algorithm in Cryptography and Network Security | Easiest Explanation for Students*. YouTube, 2020. *YouTube*, www.youtube.com/watch?v=CxAfmNQG0yk. Accessed 10 July 2021.

Stine, Kevin, and Quynh Danh. "Encryption Basics." *NIST*, National Institute of Standards and Technology, 2 May 2011, csrc.nist.gov/publications/detail/journal-article/2011/encryption-basics. Accessed 21 Sept. 2021.

Torres, Jessica. "Cryptography: Symmetric vs Asymmetric Encryption." *Gitconnected*, 18 May 2020, levelup.gitconnected.com/cryptography-symmetric-vs-asymmetric-encryption-db36277c8329. Accessed 15 Sept. 2021.

Ward, Mark. "How the Modern World Depends on Encryption." *British Broadcasting Corporation*. *BBC News Services*, www.bbc.com/news/technology-24667834. Accessed 14 Sept. 2021.

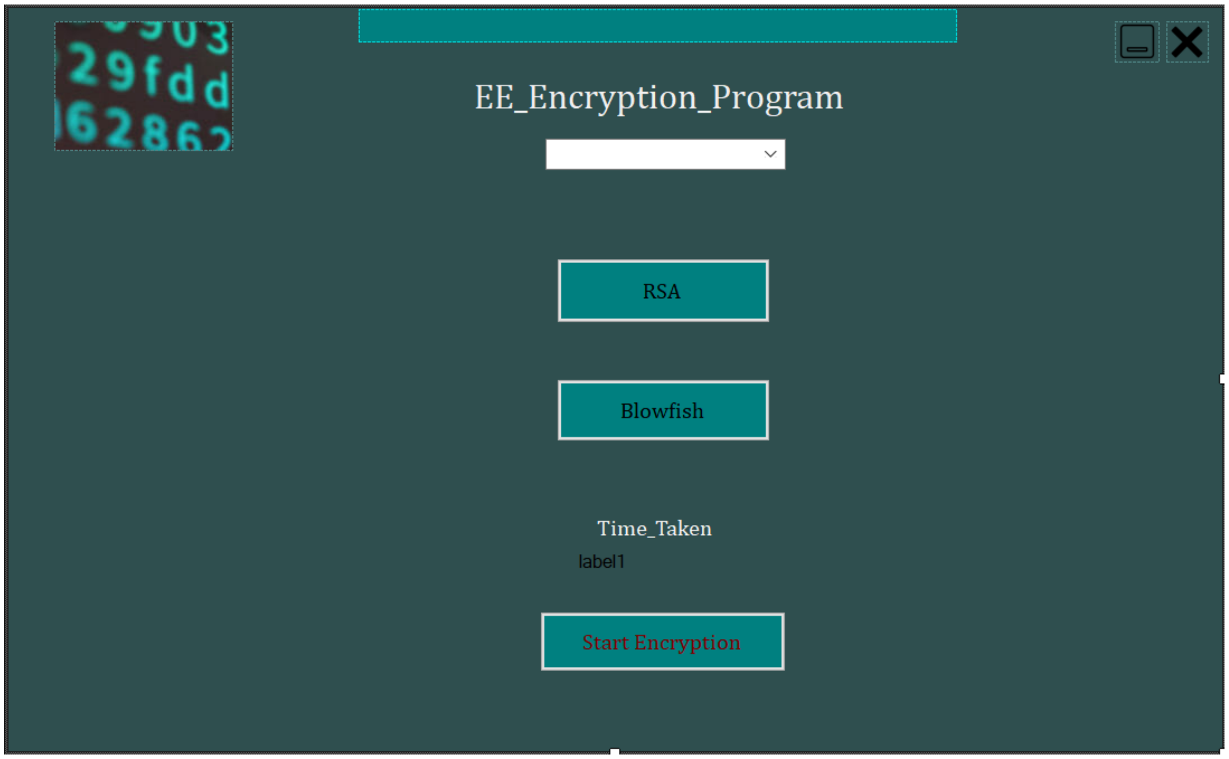
"What Is the Complexity of Bitwise Operations?" *CodeForces*, 13 Jan. 2014, codeforces.com/blog/entry/10337. Accessed 20 Sept. 2021.

10 Appendices

Appendix A: Code for the Program

```
using System;
using System.Windows.Forms;
using Org.BouncyCastle.Crypto.Generators;
using Org.BouncyCastle.OpenSsl;
using Org.BouncyCastle.Crypto.Utilities;
using Org.BouncyCastle.Crypto.Engines;
using System.IO;
using System.Security.Cryptography;
using System.Diagnostics;
using Org.BouncyCastle.Crypto;
using Org.BouncyCastle.Security;
using Org.BouncyCastle.Crypto.Parameters;
using Org.BouncyCastle.Crypto.Encodings;
using Org.BouncyCastle.Crypto.Paddings;
using Org.BouncyCastle.Utilities.Encoders;

namespace EE_EncryptionProgram
{
    public partial class MainInterface : Form
    {
        Boolean[] options = new Boolean[1];

        OpenFileDialog openFileDialog = new OpenFileDialog();

        private const int bitSizeRSA = 2048;

        private const int RSACipherBlockSize = 200;

        private const int bitSizeBlowfish = 64;

        string fileLocation = "";

        private Boolean mouseDown;

        Stopwatch stopwatch = new Stopwatch();
        private int mousex;

        private int mousey;

        byte[] result = null;

        Boolean RSAEnabled, BlowfishEnabled;

        string outputTextFile = String.Empty;

        public MainInterface()
        {
            InitializeComponent();
        }


        private void optionsEncryption_DropDownClosed(object sender,
EventArgs e)
        {
            int identifierEncryption = optionsEncryption.SelectedIndex;
            try
            {
                switch (identifierEncryption)
                {
                    case 0:
                        openFileDialog.Filter = ("text files(*.txt)|");

                        if (openFileDialog.ShowDialog() ==
System.Windows.Forms.DialogResult.OK)
                        {
                            fileLocation = openFileDialog.FileName;
                            outputTextFile =
@"C:\Users\Admin\OneDrive\Documents\EE_Output_Data\TextOutput";
                            options[0] = true;
                        }
                        break;
                }
            }
            catch (Exception ex)
            {
                Console.WriteLine(ex.ToString());
            }

            startEncryption.Visible = true;
        }

        private void pictureBox2_Click(object sender, EventArgs e)
        {
            Application.Exit();
        }

        private void pictureBox3_Click(object sender, EventArgs e)
        {
            this.WindowState = FormWindowState.Minimized;
        }

        private void pictureBox4_MouseDown(object sender, MouseEventArgs e)
        {
            mouseDown = true;
        }

        private void pictureBox4_MouseMove(object sender, MouseEventArgs e)
        {
            if (mouseDown == true)
            {
                mousex = MousePosition.X - 225;
                mousey = MousePosition.Y - 2;

                this.SetDesktopLocation(mousex, mousey); //Sets the desktop
location to be the mouseX and mouseY coordinates
            }
        }

        private void pictureBox4_MouseLeave(object sender, EventArgs e)
        {
            mouseDown = false;
        }

        private void EncryptFileRSA(string filePath)
        {
            byte[] finalOutput = null;

            try
            {
                byte[] data = System.IO.File.ReadAllBytes(filePath);

                RsaKeyPairGenerator rsaKeyPairGenerator = new
RsaKeyPairGenerator();
                rsaKeyPairGenerator.Init(new KeyGenerationParameters(new
SecureRandom(), bitSizeRSA));
                AsymmetricCipherKeyPair keyPair =
rsaKeyPairGenerator.GenerateKeyPair();

                RsaKeyParameters PublicKey =
(RsaKeyParameters)keyPair.Public;

                IAsymmetricBlockCipher cipher = new OaepEncoding(new
RsaEngine());
                cipher.Init(true, PublicKey);

                stopwatch.Start();

                int dataLength = data.Length;

                for (int i = 0; i < data.Length; i = i + 1)
                {

                    finalOutput = cipher.ProcessBlock(data, 0,
RSACipherBlockSize);
                  dataLength = dataLength - 200;
                }

                stopwatch.Stop();

                timeDuration.Text =
stopwatch.Elapsed.TotalMilliseconds.ToString();

                stopwatch.Reset();

                StreamWriter streamWriter = new StreamWriter(fileLocation);

                streamWriter.WriteLine(finalOutput);

            } catch (CryptographicException c)
            {
                Console.WriteLine(c.Message);
            }

        }

        private void EncryptFileBlowfish(string filePath)
        {
            try
            {
                byte[] data = System.IO.File.ReadAllBytes(filePath);

                BlowfishEngine engine = new BlowfishEngine();

                PaddedBufferedBlockCipher cipher = new
PaddedBufferedBlockCipher(engine);

                CipherKeyGenerator cipherKeyGenerator = new
CipherKeyGenerator();

                cipherKeyGenerator.Init(new KeyGenerationParameters(new
SecureRandom(), bitSizeBlowfish));

                KeyParameter blowFishKeyParameter = new
KeyParameter(cipherKeyGenerator.GenerateKey());

                byte[] outputData = new
byte[cipher.GetOutputSize(data.Length)];

                cipher.Init(true, blowFishKeyParameter);

                stopwatch.Start();
                cipher.ProcessBytes(data, 0, data.Length, outputData, 0);
                stopwatch.Stop();

                timeDuration.Text =
stopwatch.Elapsed.TotalMilliseconds.ToString();

                stopwatch.Reset();



            } catch (CryptographicException c)
            {
                Console.WriteLine(c.Message);
            }
        }

        private void RSA_Encryption_Click(object sender, EventArgs e)
        {
            BlowfishEnabled = false;
            RSAEnabled = true;
        }
        private void Blowfish_Encryption_Click(object sender, EventArgs e)
        {
            RSAEnabled = false;
            BlowfishEnabled = true;
        }

        private void startEncryption_Click(object sender, EventArgs e)
        {
            if (BlowfishEnabled == true)
            {
                EncryptFileBlowfish(fileLocation);
            }

            if (RSAEnabled == true)
            {
                EncryptFileRSA(fileLocation);
            }
        }

    }
}
```

Appendix B: User interface for the GUI



Appendix C: Raw Data of times obtained

C1: Raw and Average Times for RSA encryption algorithm

Data-Size(KB)	10	20	30	40	50	60	70	80	90	100
Trial 1	151.1588	217.5902	347.3237	448.3352	1077.5307	1168.7091	1090.2477	1395.2201	1575.9049	1989.8048
Trial 2	151.1588	177.0235	339.35	454.4357	811.6421	1217.7013	1555.9948	1620.925	1523.1873	1765.5393
Trial 3	151.1588	210.4865	339.1554	534.1642	557.4329	948.8389	1131.8158	1551.2303	1501.378	1887.4394
Trial 4	151.1588	268.5833	383.6824	774.3345	994.0378	969.9079	1151.2158	1536.5786	1403.9741	1889.9275
Trial 5	151.1588	281.7217	349.5405	782.7353	790.7275	678.4882	1002.3408	914.7717	1577.7077	1748.5754
Trial 6	151.15	221.83	364.55	502.94	881.49	1297.0	1318.4	1426.2	1562.324	1988.6

	88	91	14	27	26	396	609	356	5	518
Trial 7	151.1588	190.1823	316.6554	676.419	878.3881	1286.6631	1159.5854	1677.6818	1764.6684	1990.7627
Trial 8	151.1588	234.5278	326.6562	428.6857	673.6593	796.921	1285.7766	1605.7174	1572.345	1721.3603
Trial 9	151.1588	229.9274	369.831	664.8188	608.5934	988.0849	1090.682	1595.1481	1853.7899	1868.8677
Trial 10	151.1588	217.8849	395.6007	697.7478	897.652	1292.3619	1293.642	1440.3801	1993.9081	1378.1992
Average	151.1588	224.97667	353.23467	596.46189	817.11564	1064.47159	1207.97618	1476.38887	1632.91879	1822.91281

C2: Raw and Average Times for Blowfish encryption algorithm

Data Size (KB)	10	20	30	40	50	60	70	80	90	100
Trial 1	0.2135	0.4253	0.6364	0.8562	1.019	1.2561	1.4787	1.6906	1.9553	2.1437
Trial 2	0.2134	0.3481	0.6361	0.6946	1.3144	1.291	0.9333	1.925	1.9517	2.2233
Trial 3	0.212	0.4247	0.6369	0.8454	1.4986	1.0423	1.4897	1.6025	1.9527	1.3886
Trial 4	0.2134	0.4254	0.6424	0.8542	1.0128	1.2897	1.8466	1.1582	1.3819	2.1649
Trial 5	0.2253	0.4234	0.6473	0.8538	0.8707	1.348	1.33	1.7041	1.9494	1.554
Trial 6	0.2134	0.4303	0.6371	0.8915	1.0658	1.2872	1.4836	1.7042	1.8767	2.233
Trial 7	0.2165	0.4254	0.5038	0.7178	1.4259	1.635	1.484	1.7354	1.5326	1.909
Trial 8	0.2119	0.4222	0.637	0.598	0.6567	0.7637	1.0673	1.707	1.9645	2.1372
Trial 9	0.1748	0.4332	0.6399	0.8524	0.9348	1.2953	1.489	1.7382	1.9506	2.1207
Trial 10	0.2122	0.4259	0.6403	0.8536	1.095	1.5504	1.4914	1.6925	1.9577	2.1203
Average	0.21064	0.41839	0.62572	0.80175	1.08937	1.27587	1.40936	1.66577	1.84731	1.99947

Appendix D: Controlled Variables

Variable	Description	Specifications (if any)
Computer and Operating System	This experiment will be run on my personal laptop:	**Processor:** 1.80 GHz Intel Core i7-8565U

	Thinkpad E490s	**Memory**: 8GB DDR4 2400 MHz
Integrated Development Environment used	The program will be run in the Visual Studio IDE only	**IDE:** Visual Studio 2019 **C# Runtime Environment:** .NET Framework 4.7.2 **C# Virtual Machine:** Common Language Runtime
Same algorithms used	The algorithm's code from Appendix A will be used.	**RSA Key Bit Size:** 2048 bits **RSA Cipher Block Size (constant):** 200 bits **Blowfish Key Bit Size:** 64 bits **Blowfish Cipher Block Size (constant):** 64 bits
Same methods called	The functions "EncryptFileRSA" and "EncryptFileBlowfish" will be called for every set of data being tested.	
Same data type used	The program will only use the STRING data for all the text files being tested.	