National College of Ireland

Time efficient factorisation of RSA semiprime numbers ©

MSc Research Project
Cybersecurity

Olwen Brangan
Student ID: x20216866

School of Computing
National College of Ireland

Supervisor: Mr. Ross Spelman

National College of Ireland

MSc Research Submission Sheet

School of Computing

Student Name: Olwen Brangan

Student ID: X20216866

Programme: Cybersecurity **Year:** 2024

Module: MSc. Research Project

Supervisor: Mr. Ross Spelman

Submission Due Date: 12 August 2024

Research Title: Time Efficient Factorization of RSA Semi-prime Numbers

Word Count: ..... **Page Count** .....

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this research. All information other than my contribution will be fully referenced and listed in the relevant bibliography section at the rear of the research. ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use another author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: Olwen Brangan

Date: 12 August 2024

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each research (including multiple copies)	□
Attach a Moodle submission receipt for the online research submission to each research study (including multiple copies).	□
You must ensure that you retain a HARD COPY of the research, both for your reference and in case the research is lost or mislaid. It is not sufficient to keep a copy on a computer.	□

Assignments that are submitted to the Programme Coordinator's Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Time efficient factorisation of RSA semiprime numbers

Olwen Brangan
X20216866

Abstract

A puzzle that remains unresolved for many years is how to factorise large semi-prime numbers used in the RSA algorithm in ***polynomial time***. Ransomware and processes that involve key exchanges and digital signatures use the RSA algorithm. A ransomware attack, particularly in the operational technology (O.T.) industry, can have terrible consequences. Therefore, it is crucial to test the RSA algorithm continually. Because of this, it is necessary to establish a solution for obtaining a decryption key should a business come under attack. Although there are often additional security measures in place, any weakness in the chain may result in the downfall of a company. Many researchers have achieved factorisation of large semi=prime numbers. However, there is ongoing regular reporting from researchers about the inability to factorise these semi-prime numbers in the same time frame regardless of the size of N. The DERMOT solution presented here quickly eliminates many numbers as prime factors of a semi-prime number. The factorisation of a 100-digit number was attempted and the first six digits matched correctly. Using this algorithm significantly reduces the range of numbers to be tested and can be easily implemented with Python.

Keywords:
RSA, Factorisation, Operational Technology (O.T.), Polynomial Time, Decryption, Ransomware.

1 Introduction

According to Erwin Schrödinger, *"The task is not so much to see what no one has yet seen but to think what nobody has yet thought, about that which everybody sees"* (Pech, 2006). This quote recalls an unresolved problem that many have encountered but which remains unresolved. It is not the factorisation of large semi-primes that perplexes the cryptography community. Instead, it is the inability to factorise large semi-prime numbers into their prime factors ***in polynomial time***. But before delving into the details of achieving such a task, the background to this problem and its importance in cryptography is discussed.

1.1 Background

One popular asymmetric (public-key) cryptographic algorithm is RSA, named after its three founders, Ron Rivest, Adi Shamir, and Len Adleman (Easttom, 2021). Integer factorisation of large semi-prime

numbers into two prime factors forms the basis of the RSA algorithm. The complexity of the RSA algorithm is due to its ability to act as a one-way function (Katz and Lindell, 2020). In other words, it is easy to multiply two numbers together but considerably more challenging to determine what those two numbers are when only provided with their product. (Wang, 2022) (Sabani *et al.*, 2022) (Nemec *et al.*, 2017), (Park, H.B. *et al.*, 2021) (Easttom, 2021), (Katz and Lindell, 2020), (Joshi et al., 2023).

The importance of the RSA algorithm is due to its role in the key exchange and authenticating processes (Park, H.B., Sim, B.Y., and Dong-Guk Han, D.G. 2021) (Katz and Lindell, 2020). (The mathematical processes of generating keys for key exchange and signing messages, together with worked examples, are described in-depth in sections 4.1 and 4.2 in the configuration manual). To understand how key exchange works, consider a hypothetical situation. Suppose Aoife wants to send Seán an encrypted message. Seán generates a public key and a private key. Aoife obtains Seáns public key and uses this key to encrypt the message. Aoife then sends the message to Seán. Seán decrypts the message with his private key. The advantage of this process is that only a person with access to Seán's private key can read the message. Unlike the RSA key exchange process used in encryption schemes, the public key is needed for verifying signatures while the private key is used for signing. In this situation, if Aoife wants to send Seán a message she generates the RSA public key and private keys. Aoife sends her public key to Seán and calculates the signature (s) by encrypting a message (x) with her private key. Aoife appends the message x with the signature s and sends the appended message to Seán. Seán receives the signed message and decrypts it using Aoife's public key.

The RSA public-key cryptosystem is computationally slow and expensive but widely used. The RSA algorithm is considerably slower than symmetric algorithms and therefore is often used in conjunction with a symmetric algorithm such as Advanced Encryption Standard (AES) which is used in banking. Ultimately the overall purpose of RSA is to ensure the confidentiality, integrity, availability, and authenticity of data.

1.2 Motivation for this research and potential beneficiaries

Weak encryption algorithms can lead to ransomware attacks, but sometimes ransomware uses the RSA algorithm. When businesses suffer a ransomware attack, it's not just the business that suffers. Our nation loses considerable finances. According to the IBM X-Force Threat Intelligence Index Report 2024, the most frequent ransomware detected by X-Force were Blackcat, CLOP, Lockbit, Blackbasta, and Royal (IBM, 2024). These ransomwares use the RSA algorithm as part of the encryption process (MSSP Research Lab, 2024), (SentinelOne, 2024), (Santos, Bunce, and Galiette, 2023), (Stephen, 2023), (Elsad, 2022), (Emisoft, 2021) (Cyware, 2019). Creating an algorithm that can factorise semi-prime numbers into their prime factors provides an opportunity to recover the decryption key when impacted by a ransomware attack.

Another motivational factor is the various people which could benefit from such a solution. This research could benefit anyone requiring a decryption key, such as law enforcement or businesses dealing with a ransomware attack. Creating an algorithm that allows the factorisation of large semi-primes means businesses that fall victim to a ransomware attack can regain access to their data and systems and minimise downtime. Researchers in academia whose work requires accuracy and precision, can use this process when dealing with small calculations or calculations of significant size. Other beneficiaries may include any business interested in testing its security. It may also be of particular

interest to those obliged to tighten security measures in preparation for a new European Union (E.U.) regulation, which comes into effect on January 17th, 2025. The new legislation called the Digital Operational Resilience Act (DORA) requires financial institutions and their third-party service providers to have policies that guarantee data confidentiality, integrity, availability, and authenticity (CIAA). By performing tests using the prime factors of N, businesses can determine the implications of using RSA numbers of various lengths concerning the encryption scheme that a company has in place. This solution may benefit those in the operational technology (O.T.) industry who prefer not to update their software due to possible implications with O.T. equipment. The O.T. industry can use this solution as part of a recovery simulation exercise. Use cases from business and academic perspectives are provided in section 6.2 of this document.

1.3 Gap in literature and contribution to the scientific literature

Although researchers have studied the RSA algorithm extensively, there appears to be a lack of research content about demonstrating computationally efficient ways for integer factorisation (Wang, 2022). Semi-prime numbers comprising up to two hundred and fifty decimal digits have been factorised, although outside of polynomial time (Katz and Lindell, 2020). In polynomial time, the computation steps required to factorise N into its semi-prime factors grow at a predictable rate rather than exponentially. This research aims to contribute to the scientific literature by using a new approach to reduce computational time for factorisation of large semi-prime numbers and discuss applications in the O.T. industry.

1.4 The research question

The aim of this research is to design a software solution with a fast factorisation time of semi-prime numbers and discuss how someone can use it in the O.T. industry. Creating a unique algorithm that can factorise large semi-primes, resulting in a faster decryption time, is the primary goal of this research. The main objective is to test the theory that there are alternative approaches to resolving the problem of factorising a semi-prime number of 100 digits or more in polynomial time. Specifically, the research aims to answer the **question**: How can a decryption key be recovered quickly during a ransomware attack that employs the RSA algorithm*?* There were three main research objectives:

Objective 1: Create a unique algorithm to reduce the computations required to factorise a semi-prime number.
Objective 2: Implement the algorithm with suitable software.
Objective 3: Provide use cases to demonstrate how someone can apply the solution in the O.T. industry.

1.4.1 Assumptions

The research assumes that sources of information on previous RSA factorisations have published the correct prime factor. The accuracy of the results was validated using multiple sources and software tools.

1.4.2 Inclusion and Exclusion Content.

A list of requirements is provided in section 1 in the configuration manual. This forms the inclusion criteria. The algorithm provided in this paper excludes the process of changing the binary form of N to an equivalent decimal number. Well-known factorisation methods and the use of a quantum computer are also excluded.

1.5 Structure of the report

Section 2 looks at a review of the literature. Section 3 describes the research methodology for obtaining suitable information and the evaluation process to determine if the proposed solution is appropriate. Section 4 contains the design specification and a description of the algorithm. Section 5 discusses the software used and development of the solution. Section 6 includes the most relevant results in support of the research question and the main objectives.

2 Literature Review

2.1 The RSA algorithm

The RSA algorithm uses a public key for encryption and private keys for decryption (Easttom, 2021). Section 4.1 in the configuration manual contains an example of calculating the private and public keys. The algorithm provided in the configuration manual can be applied when one person wants to send another a message. For example, suppose Aoife intends to communicate with Seán; Aoife needs Seán's public key to do this. Seán then uses his private key to decrypt the message.

Generation of the encryption keys involves multiplying two large prime numbers, p and q, which results in N, a semi-prime number. Phi n is then calculated to establish how many numbers from one to N are coprime (two numbers with no common factors, e.g. eight and nine) to N. Phi n is commonly known as Euler's totient function. For prime factor p, this is one less p because p is a prime number with only two prime factors: itself and one. Therefore, all other numbers from one to ‘p minus one’ are coprime. The same applies to prime factor q. The public exponent (encryption) key (e) is then selected and must satisfy the rule that the greatest common denominator of e and phi n equals one. The private (decryption) key is then calculated using the formula $d \times e \equiv 1 \bmod \Phi(n)$.

2.2 Factorisation of semi-prime numbers

2.2.1 The RSA factoring challenge

RSA encryption and decryption are based on modular exponentiation, and obtaining the prime factors of N has remained an enigma until now. In 1991, the RSA factoring challenge was established to encourage research in computational number theory. Participants were offered monetary prizes to promote research in this field. The smallest semi-prime provided (RSA-100) consisted of one hundred decimal digits and was successfully factorised[1]. Some methods used to factorise RSA semi-primes successfully were the quadratic sieve and general number field sieve.

[1] https://www.rsa.com/

These methods are not used in this research and are not discussed here. What is relevant to this research is that the methods used were time-consuming, so another solution was required. To put this statement into context, it took Benjamin Moody 73 days in 2009 to factorise a 512-bit key (Easttom, 2021). In 2020, the RSA-250 semi-prime number was factorised into its prime factors using the number field sieve algorithm, and it took approximately 2700 CPU core years.

2.2.2 Squaring N

Many researchers recently proposed various factorisation methods for RSA semi-primes. In 2021, Overmars and Venkatraman proposed a solution for factorising large semi-prime numbers (Overmars and Venkatraman, 2021). The authors demonstrated that semi-prime numbers were factorised using the Brahmagupta-Fibonacci principle to reduce four Pythagorean quadruples to two triples. A Pythagorean quadruple consisted of four integers in total: three unequal integers and semi-prime number N. Each integer was squared, and the sum of the three integers squared resulted in N squared $(g^2 + h^2 + i^2) = N^2$. The authors successfully showed that the sum of three squares resulted in the factorisation of a semi-prime. However, a limitation to this method was that the number of viable solutions increased as N increased. One such case study provided by the authors was N, which is equal to a three-digit number. Overmars and Venkatraman discovered forty-five possible sums of squares with eleven possibilities, leading to the factorisation of the semi-prime. Whilst this was a solution for small N, RSA sometimes involves semi-primes consisting of 4096 binary digits. Overmars and Venkatraman's work lies in deep contrast with the fundamental basis of my solution. Interestingly, their methodology involves the squaring of N, which I think is problematic due to the issue of storing large numbers. For this reason, a different approach is used in this research, which involves the immediate reduction of N.

2.2.3 The reduction of N

Inan used a method to approximate RSA prime factors using the square root of N (Inan, 2022). The advantage of this approach was that it resulted in the immediate reduction of N. Inan found the distance of p from the square root of N (ΔX) and the distance of q from the square root of N (ΔY). The distance in the ΔY direction was 5% to 15% longer than in the ΔX direction, and so ΔX was investigated further. ΔX, in this case, was the distance from the square root to the bigger prime factor of N. Inan ran the program multiple times and calculated the mean value of each run. Results showed a high probability of correctly matching the first to the fourth digit, but the disadvantage of this approach was that it was rare to match beyond the fifth digit. In my opinion this was a helpful method because one of the primes will always be less than or equal to the square root of N. For example, consider two primes, p and q. Let p equal 13 and q equal 17. Their product N is equal to 221. The square root of 221 is 14.87. p, which has a value of 13, is less than the square root of N (14.87). Now, consider two numbers with a more extensive range between them. For example, p is equal to 307, and q is equal to 701. N is equal to 215207. The square root of N is equal to 463.90. p has a value of 307, which is less than the square root of N. This method was a practical starting point for achieving the objectives of this research.

Next, consider the length of the square root of N, which is approximately half the number of digits of N. For RSA 250, the square root is a 125-digit number. In the case of RSA-155, the semi-prime was factorised into two prime numbers consisting of 78 digits. Successful factorisation was achieved, although the method used took 3.7 months, using 300 computers (Cavallar et al., 2000). Nonetheless,

two essential facts have been established. First, one of the prime factors of N will always be less than or equal to the square root of N, and second, the length of p and q will always be half the length of N or close to half the size of N.

2.2.4 Software for implementing a solution.

Another consideration for the design process was what software to choose for implementing the solution. Rodriguez, Castang, and Vanegas (2021) chose Python as their software choice and imported the time library to evaluate the processing time of an RSA algorithm. The authors implemented their algorithm on a Windows 64-bit operating system containing an Intel core i5-10210U central processing unit and eight gigabytes of random-access memory. The prime numbers tested were four, eight, and ten bits, so the processing time was also small. A memory overflow occurred when using prime numbers consisting of 12 bits. Flynn (2021) also obtained overflow errors in Python when using logarithms (logs) and semi-prime factorisation in Python. The author also found that too many iterations were required using the log method. The decryption key was not calculated using this method.

However, it is essential to remember that logs were initially generated to make calculations easier. Do not underestimate the power of logs. Bellos (2020) reminded readers that logs were the reason Johannes Kepler, a German astronomer, could calculate Mars's orbit. The formula for calculating the log of A multiplied by the log of B is equal to the log of A plus the log of B. This provides an opportunity to simplify multiplication problems into addition. To apply the equation to find the prime factors of a semi-prime number, let A equal p (prime one) and B equal q (prime two). The log of p and the log of q added together results in the log of N. Using logs can be beneficial for obtaining an initial approximation. However, the disadvantage of using logs is that they increase the number of steps required to obtain prime factors. It is for that reason that it is better to perform calculations in decimal form.

It is not Python that is the issue (certainly for factorising semi-prime numbers less than one hundred digits) but the strategy that has been applied. Python[2] is a very powerful language. It is very useful for performing calculations. It does have some issues, for example, not immediately returning an answer when factorising a 10-digit number. However, the Decimal module in Python allows the multiplication of two possible prime factors to determine if the result is the original semi-prime number.
Also, instead of trying to obtain all factors for a number in Python, many numbers can be eliminated first. For example, all even numbers are not prime. Obtaining the first few digits of p and q will eliminate another large portion of numbers.

2.3 Primality tests

Once factors have been determined, the next step is to ensure that the factors are prime numbers. Many methods exist for testing whether a number is a prime number or not. Eratosthenes sieve (Easttom, 2021) is an accurate way to determine if a number is prime. Using this method, all numbers that are divisible by two, three, five, and seven are eliminated, and the remaining numbers are prime. This is too impractical, though, when dealing with large numbers. Too many computations are required.

[2]https://www.python.org/

Another well-known technique for primality testing is Fermat's little theorem. Consider an integer n. p is prime if n^p minus n returns an integer (Easttom, 2021). Nowadays, however, there are useful software tools available, such as tools from Wolfram Alpha[3] or SageCell[4]. These software tools can handle numbers of significant sizes.

2.4 Summary

For many years now, those who have analysed RSA numbers have considered it computationally infeasible to factorise semi-prime numbers of 100 digits or more into their prime factors in polynomial time. Two main tasks are necessary to find prime factors: determine the factors of the semi-prime number (factorisation of the semi-prime number) and determine if the factors obtained are prime or not (primality tests). Both tasks have been achieved, but according to many authors, ***not in polynomial time***. Various methods have been tried and tested, including the squaring of N, the initial reduction of N, and the use of logs. Python is a popular choice for the implementation of solutions, but issues such as memory overflow errors often occur. Online tools such as Sage Math or WolframAlpha are available to check if a number is prime or not.

Two important facts were established when reviewing the literature: one prime will always be less than or equal to the square root of N, and the number of digits for p and q is often half or close to half of the number of digits contained in N. This is a good starting point for establishing a solution.

3 Research Methodology

The research methodology was evaluated based on how the approach taken met the research objectives.

3.1 Requirements Analysis: Methodology

To write the plan, I first wrote a list of requirements. (The requirements specification is provided in Section 1 in the configuration manual). Once a gap in research was found and the research objectives decided, the requirements list was updated to include additional requirements. For example, the list was updated to include software that can perform computations with large numbers.

3.2 Requirements Analysis: Evaluation

The requirements list was assessed by ensuring the plan for the solution included the pre-determined requirements and any additional requirements necessary to complete the task.

3.3 Research Question and Objectives: Methodology

Cryptography was selected as the broad topic to focus on. Then, a list of possible sources of where to find information on Cryptography was written.

[3]https://www.wolframalpha.com/
[4]https://sagecell.sagemath.org/

Sources of information included journals from reputable databases, cryptography books, and company websites available to the public, including annual threat reports from cybersecurity organisations. Each database was searched according to various search terms (examples of searches are provided in Appendix A in the configuration manual). Folders with the different database names were created and saved on My Desktop. All journals of interest found in a particular database were saved into the folder with its matching database name (See Section 2 in the Configuration manual). The abstracts of the selected journals were read, and papers containing relevant abstracts were selected. These papers were then read in further detail while searching for future work or gaps in research. The journals deemed useful were put into a folder named Useful on My Desktop. At this point, gaps that had the potential to be measured and implemented as an ICT solution were considered. Then, a mind map of things that can be measured was created. (Refer to section 3 in the configuration manual).

One gap that arose consistently throughout the literature from researchers in various disciplines was the inability to factorise RSA semi-prime numbers ***in polynomial time*** on a single modern computer. Having decided to focus on this topic, areas where this research could be useful were identified. As a result, the focus of the research was narrowed to focusing on ransomware in the O.T. industry. This led to the formation of a research question: How can a decryption key be recovered quickly in the event of a ransomware attack that employs the use of the RSA algorithm*?* This research focuses specifically on the factorisation of semiprime numbers to gain advantage in the decryption process. The ethics of this research question and the essential requirements of the research were considered. Having established that the research question satisfied the main requirements of the project, three essential aspects of the research were established, which formed the research objectives.

Once the objectives were decided, a list of tasks and subtasks needed to achieve each objective was constructed. For example, for research objective one, there were two main tasks: create a unique algorithm and factorise a semi-prime number using a single computer for both tasks. There are two sub-tasks involved in factorising a semi-prime: finding factors of a very large number and determining if these factors are prime numbers. These tasks were added to the list of requirements in the configuration manual.

The best approach to selecting a suitable semi-prime number to factorise is to select one that had already been factorised. This was the approach taken to determine if the proposed algorithm would work. The factors obtained could be compared with the factors obtained from the known factorised semi-prime number. By entering 'RSA factors' into a Google search engine, the ‘RSA Factoring Challenge’ with a list of RSA semi-prime numbers and their factors was discovered. The smallest RSA semi-prime number provided with its prime factors was RSA-100, a 100-digit number. This was selected as my starting point and added to the requirements list. However, the selection of a 100-digit number is for an initial assessment of the suitability of the proposed algorithm. It is important to establish if the algorithm works for a 100-digit number before attempting a much larger number. In terms of the research objectives, selecting a 100-digit semi-prime number is suitable for implementation with software. It is important to note that many semi-prime numbers nowadays are using numbers containing more than 100 digits.

An exploration of the literature was then carried out to determine factorisation methods and primality tests that have been used to avoid the repetition of previous attempts. The requirements formed the

inclusion criteria. Previously used methods of factorisation formed the exclusion criteria as well as the process for changing the binary semi-prime number to its digital equivalent.

3.4 Research question: Evaluation

The research question was evaluated in terms of the research requirements. The initial question considered was: Is this research ethical? It is important that hackers cannot solve this factorisation problem. However, it is essential for this reason that this research is ongoing to demonstrate that alternative solutions are put in place immediately. There is also an additional benefit to those affected by ransomware. If this factorisation can be achieved, then the process of obtaining a decryption key may be accelerated. Is the research based on a genuine problem that exists? The answer is yes because this problem has been attempted by many researchers over a long period. To make the research unique, I decided to focus on a solution that can be applied to the O.T. industry. Is the research based on an ICT solution, and is it measurable? Yes, the algorithm can be implemented in Python. The measurement of interest is how many steps are required to factorise a semi-prime number. Is it software that can be used in the O.T. industry? Yes, Python can be used on multiple operating systems and has an unofficial build for Android and iOS. Does the proposed work make a significant contribution to research? The solution provides an alternative approach to solving a problem long pursued by researchers. The approach taken is discussed in terms of balancing equations. Does the research cover aspects from various disciplines? Yes, the research includes Cryptography, Mathematics, responding to a ransomware incident in the O.T. industry, Computer Programming, and Research in Computing. The algorithm is a unique combination of decimals and logs which can be implemented on a single computer. The method for obtaining results (the DERMOT algorithm) is explained in detail and can be easily reproduced by other researchers so the results can be verified.

The process from collecting data to analysing data and obtaining results is discussed in-depth. Any person with access to a regular computer can easily reproduce the results via Python. The main requirement of the research was to create a unique solution to factorise large semi-prime numbers, which may assist in the fast decryption of RSA-based ransomware in the O.T. industry. Using results from the RSA factoring challenge meant that the accuracy of results from the algorithm proposed here was verified.

3.5 Research Objective 1: Evaluation

Time to respond to an incident is most important when a ransomware attack is carried out on the O.T. industry. This is because systems that are impacted may result in serious injury or death. Therefore, recovering a decryption key should be done as quickly as possible. Obtaining the decryption key is possible if one of the prime factors (p or q) used to obtain semi-prime number N is known. Therefore, the first objective was to design a unique algorithm that minimises the number of computations required for the factorisation of a semi-prime number. A search for RSA factors was performed to obtain an appropriate size semi-prime number. This resulted in finding the RSA factoring challenge. As the smallest number provided was 100 digits, this number was chosen as the minimum number to test.

3.6 Research Objective 2: Evaluation

To achieve the second objective of implementing an Information and Communications Technology Solution (ICT) solution with suitable software, a search of the literature review was performed. This led to the discovery of various software applications that were used by researchers. A Google search using the words 'software for calculating large numbers' was also conducted. Python, R and Sage Math were chosen to investigate further. Each of these were compared with each other (Refer to Section 5 in the configuration manual).

Calculations with R were completed due to some familiarity with this programming language. A table of prime numbers (refer to section 5.1.2 in the configuration manual) was generated, but limitations were found when calculating larger numbers (refer to section 5.1.6 in the configuration manual). Other software was investigated, and three other potentially useful options were found: namely Python, Sage Math, and Wolfram Alpha. Sage Math builds on R, Python, and other software and is very powerful and suitable for tasks dealing with calculations involving large numbers.

However, I chose to use Python because many researchers had previously used Python, and it was important to be able to verify claims made by other researchers. Also, Python is free to download and has lots of documentation. Having decided to test Python's suitability, three necessary tasks were recognised. First, establish if Python is easy to use and easy to learn. This was accomplished by writing code to do some calculations. Indeed, Python is easy to use, easy to learn, readable, and can be easily maintained. Second, software was checked to ensure that calculations with numbers of at least 100 digits (the smallest RSA semi-prime number factorised and published on the Wikipedia website) could be accomplished. This was found to be the case. Third, to ensure the reliability of the software, the accuracy of computations was verified. While performing many calculations with Python, some issues were discovered, but there were also ways to overcome those issues. For example, the Decimal module is required to maintain the accuracy of numbers. To test the accuracy of the software, the log of various numbers was calculated and then their corresponding inverse logs to ensure that the original numbers were obtained.

To validate the results, the capabilities of online calculators were also investigated to ensure accuracy and precision were not affected during computation. What was found was most online calculators cannot return accurate results when performing computations with numbers of 100 digits or more. However, one big number calculator[5] that was tested was found to be useful (section 5.5 in the configuration manual). To test its capabilities, a 100-digit number was squared. This calculator was used to compare results obtained by also squaring a 100-digit number in Python with results obtained using the big-number calculator. The same result was returned. However, further rigorous testing is required for significantly larger numbers than those containing one hundred digits. Selecting a second software tool minimised the likelihood of error and verified the validity of the results. Satisfied that Python is sufficient for achieving the goals of this research and due to its ease of use, Python was chosen for writing the algorithm. Examples of Python calculations are contained in section 5.2.1 in the configuration manual.

[5]https://www.calculator.net/big-number-calculator.html

The fastest way to determine if a factor is prime or not was then explored. The easiest and fastest solution is to use freely available software such as Wolfram Alfra or SageMathCell (section 5.3 and section 5.4 in the configuration manual).

3.7 Research Objective 3: Evaluation

Use cases were created for the algorithm to show who can benefit from this solution and how the solution can be applied.

3.8 Data Collection: Methodology

A literature review was performed using reputable sources. A folder named ‘Useful’ was created, and research papers of interest were added to this folder. Two critical tasks that needed to be done before proposing a solution were identified. First, it was essential to become familiar with the work of major contributors to the RSA algorithm. Second, a search was carried out to detect if solutions were available and avoid repetition of methods already used.

A critical analysis of related works was fulfilled to establish useful contributions that researchers have provided and establish potential gaps or improvements that are required. This provided knowledge about further avenues to explore or to abandon. The initial stage of the critical analysis began with identifying the article title, author(s), date of publication, and name of journal or conference. Focus was placed on the research objectives of the article, the methods used by the author(s), and their results in terms of the author(s) research objectives while considering the strengths and weaknesses of the approach taken. Articles were also assessed in terms of contribution to research. As each article was critiqued, it was documented in the reference section. In some cases, articles listed in the reference section of selected articles led to the exploration of other articles.

The related works were then analysed in terms of the research question presented in this document and how research objectives could be realised. For example, what software did researchers use for performing large-number calculations? Is this something that can be learned within the established timeframe requirements? Is the software freely available? Can the software be easily attained from a reputable source?

3.9 Data collection: Evaluation

To evaluate the quality of the data, journals from reputable databases were selected. Research papers that did not have well-explained methods and results were excluded. The research papers were also evaluated in terms of how they meet the research objectives. The literature review provided knowledge of various factorisation methods and primality tests, which was a necessary step towards establishing essential knowledge in this area. A critical analysis of the literature also helped to establish if the proposed solution would likely work or if similar obstacles that other researchers were presented with would be likely.

3.10 Designing the Algorithm: Methodology

To design the algorithm, it is necessary first to select appropriate software that can produce accurate results. Having selected Python as the software of choice, the software was tested to ensure accuracy and precision. The Decimal module was deemed appropriate for this task. Having determined Python's suitability, the next step was to select an appropriate method to achieve the research objectives. The best method is the one most suitable for meeting the research objectives, which is to factorise a semi-prime N in polynomial time.

Once it was determined that suitable software was available, researching how the RSA algorithm works was the next step because its purpose is likely to determine how it is designed. The first step in the standard multiplication process taught in Irish schools was considered to establish a starting point for the calculation. It begins with multiplying the last two digits of two numbers. However, these are the two numbers to be determined. Therefore, the best starting point was to figure out an approximation of the first digit of either p or q. An approximation of p and q was found as follows: first, the first two digits of N were selected. These two digits were named N2 to distinguish between the first two digits and the full one-hundred-digit number N. Then, the factors of all numbers less than or equal to N2 were established. The highest factor of N2, which can be squared and results in a number less than N2, was selected. For example, in the case of RSA 100, the first two digits are fifteen (N2). The square root of N2 was calculated to determine which of the factors found are likely to be the first digits of p and q. The square root of fifteen equals 3.87. The first digit of the square root of fifteen is three. Therefore, the number three is assigned to variable p1. Any number that could be multiplied by p1 and remain ***less than*** N2 was labelled q1. For example, three multiplied by five is 15. As N2 is equal to 15, five is not appropriate. Emphasis is placed on the words less than N2 because as more digits are multiplied, the starting digits of p and/or q are likely to increase. When p and q are approximately 50 digits long, the first digit of p or q may change. For N equal to 15, if p1 is equal to three and q1 is equal to five, then N2 may not be equal to 15 when additional digits are added. For example, if three is changed to 35 and five is changed to 52, then the first two digits of N will not be 15 because 35 multiplied by 52 equals 1820. It is likely, then, that the numbers to consider for p 1 and q1 are one less than the highest number in the set of factors obtained above. Therefore, when N2 equals 15, the first digit of q1 is likely to be four as p1 is three. Three multiplied by four is ***less than*** N2.

Now, with an approximation of the first digit of p and q, the second digit of both numbers was calculated using a process of balancing equations. For those working in the chemical industry, this is comparable to a chemical compound such as sodium chloride. The equation for the decomposition of sodium chloride into its elements is represented as NaCl -> Na + Cl. Similarly, N needs to be separated into two prime factors, p and q. Then, apply the chemistry concept: N -> p + q. In the O.T. industry, control systems with feedback loops are often employed. The methodology used here also applies a similar concept. N is starting with two factors, itself and one. Consider N to be one of these factors and assign it to variable q. The variable p then has a value of one. Now, p needs to increase to lie closer to the value of the square root. Similarly, q needs to decrease to lie closer to the value of the square root. This means that a change (output) in one variable (system) impacts the input to the other variable (system).

3.11 Designing the Algorithm: Evaluation

RSA is used as a security means. Therefore, someone selecting suitable prime numbers for use needs to determine how security can be maximised. If one prime number is considerably less than the other prime number, then security is less effective. This is easily proven because if one prime factor is known, then the second prime factor is easily obtained by dividing the semi-prime number by the known prime factor to give the second prime factor. If a person searching for prime factors tests all numbers up to the square root of N and if one of the prime factors is much less than N, then prime factorisation will be attained much faster. This does not help maximise security. Therefore, prime factors p and q are likely to be closer to the square root of N rather than farther away. Of course, this is not guaranteed. But based on this assumption, it makes sense to look for prime numbers close to the square root of N. This means that the first digit in p or q is likely to be close to the first digit of the square root of N. Of course, it may be equal to the first digit because of the size of the numbers involved. Small numbers were also factorised to ensure the approach suggested for balancing equations provides accurate results.

The decimal system was used to ensure accuracy. The process of using logs was useful for helping to establish the first few digits. However, using logs adds more computations to the process by obtaining logs and inverses of numbers.

3.12 Implementing the algorithm: Methodology

The mpmath module was imported. This was useful for functions such as calculating the square root. The decimal module was also imported as this is suitable for decimal numbers. The Decimal class was then imported from the decimal module. The getcontext function was imported to set the precision. The precision was set to a number greater than the number of digits of N.

The code was then written so that the user could input a number, which is stored in a variable named N. The RSA-100 semi-prime number was entered. The process was then repeated using a number 250 digits in length. The variable type was checked to ensure the number was stored as an integer. The initial starting point for separating the number into two factors was to assign known factors of the semi-prime. Known factors of the semi-prime number are one and the semi-prime number itself. For this reason, the numbers one and the semi-prime number were assigned to variables p and q. Each variable was then adjusted through a process of multiplication and division to achieve a point where they both had the correct first digit. The process of using logs was then used to refine the results.

3.13 Implementing the algorithm: Evaluation

To test the accuracy and precision of Python, calculations were performed, which returned a decimal number of 100 digits following the decimal point. Whole numbers were then tested that divide evenly and returned no digits after the decimal point. However, the last digit of each semi-prime number is an odd number. Obtaining the square root or dividing these numbers by two will result in a decimal number, which is not a prime number. Therefore, it is important to identify an appropriate number to divide the semi-prime number by.

A command was implemented that allows the user to input numbers of various lengths. First, the RSA-100 number was entered and assigned to a variable named N. The number of digits in the number entered was counted to ensure there was no rounding occurring. The process was then repeated for a 250-digit number. The reason a number with 250 digits was chosen was because a number this size or larger is more likely to be chosen nowadays for the RSA algorithm.

3.14 Obtaining and Analysing Results: Methodology

To achieve results, each step of the process was carefully planned to ensure that the methodology used was the best approach for achieving the research objectives. The algorithm was implemented in Python to produce outputs and obtain results. Once the semi-prime number was factorised, the number was input into SageCellMath to establish whether it was a prime number. The factors obtained were compared with the number provided on the RSA factoring challenge.

3.15 Obtaining and analysing results: Evaluation

To evaluate the methodology for obtaining and analysing results, it was necessary to ensure that the number of operations required to factorise N, and the number of tasks required to determine if p or q is prime were predictable with different values of N. This was more important than calculating the time taken to obtain a result because various factors influence time. The use of loops in programming is likely to impact this goal negatively as a loop uses a counter, and an increase in n will result in additional steps.

4 Design Specification

The first step of the process is to run Python. This can be done from the command prompt on a Windows machine.

4.1 Word-based description of the algorithm

The quickest solution is to do the calculation using the decimal system. This method is less likely to be affected by the butterfly effect due to the transformation of data. Once the prime factors are obtained, the result can be changed to binary. For example, consider the random number 15222173, which is the product of 3797 multiplied by 4009. (Note: 15222173 is not prime. It is just used for illustration purposes). Assign the number to be tested to a variable N. Here, a technique is used that allows the software to accept varying values of N. To ensure that the code does not need to be rewritten multiple times, the software script is designed so that the user is asked to enter the number to be factorised. Take the first two digits of the number N: in this case, the first two digits of N are one and five. Find the factors of fifteen other than one and fifteen. The factors of fifteen, excluding one and fifteen, are three and five. Because three multiplied by five is exactly fifteen, the highest factor is reduced by one. Now, the initial digit of p is three, and the initial digit of q is four. Assuming proper security is implemented, both factors will not be equal. The square root is the point at which both factors are equal. One factor will always lie above the square root of N, and the other will always lie below the square root of N. Therefore, obtain the square root of N to determine the maximum number for one factor (p) and the minimum number for the other factor (q).

```
>>> squareroot = mpmath.sqrt(N)
>>> squareroot
mpf('3901.5603288940695297126057500
```

As shown in the screenshot above, p lies below 3901.56, and q lies above 3901.56. To ensure accuracy, the number of digits after the decimal point is three times the number of digits contained in N.

The process for determining the second digit of p and q is described in Table 1 below. This displays the technique that underlies the implementation of the algorithm:

	p	q	
x 3	1	1522	/ 3
x 1.00657462	3	507.3333333	/1.000657462
x 1.245700246	3.00197238	507	/ 1.245700246
	3.73955773	406.9999999	
x 10	37.3955773	40.69999999	10

Table 1: Displaying the technique that underlies the implementation

The first digit of p is three, and the first digit of q is four. To obtain the second digit of p and q, take the first four digits of N (1522). Divide 1522 by three and assign it to q. Now, multiply p by 3. Now p is three, and q is 507.3333333. To obtain a whole number, divide 507.3333333 / 507 = 1.000657462. This means that 507.3333333 is divided by 1.000657462 to obtain 507. As a result, the opposite math operation is performed on p. p is multiplied by 1.000657462, which results in 3.00197238.

It was established earlier that the first digit of q is four. This means that 507 needs to be reduced to a number beginning with four. To change the first digit to four, divide 507 by 407. The result is 1.245700246. This means 507 needs to be divided by 1.245700246 to obtain 406.9999999. p is therefore multiplied by 1.245700246. p is now 3.73955773. To obtain a whole number with two digits, 3.73955773 is multiplied by 10, and 406.9999999 is divided by 10. Now, the two numbers that are remaining are 37.3955773 (p) and 40.69999999 (q).

Comparing this with the original factors 3797 and 4009, the first two digits match. The log of the semi-prime number is then calculated. This helps to prove the remaining digits because the value of p and q is now an addition exercise, but care needs to be taken when carrying over digits due to the addition of two digits exceeding ten.

5 Implementation

Note: Before implementing the algorithm, it is recommended that some commands be run first to ensure the precision is set appropriately.

The main goal of the implementation is to provide a solution that is accurate, precise, fast, and reproducible. The ideal solution is achieved in polynomial time, i.e. the number of steps required is predictable for different values of N.

5.1 Software Requirements

The algorithm was implemented in Python (Version 3.11.9), which was running on a Microsoft Windows 11 Home operating system. The machine used was an x64-based P.C. with 8.00 GB RAM and an Intel i5 core processor.

5.2 Solution Development

The modules necessary for the implementation of the algorithm include the mpmath and decimal modules. The number of digits in N dictates the minimum number at which the precision should be set. The user is requested to enter a semi-prime number to be factorised. The variable storing the semi-prime number is then checked to ensure it is of type integer. The square root of N is then calculated. Note the first two digits of N. In the case of RSA-100, the first two numbers of N are 15. The factors of the first two digits of N are obtained. This is achieved by running a for loop, which checks the numbers from one to N. If N divided by a number returns a result of exactly zero, then the number is a factor. (Note: For this to work, the code must be indented as shown in the DERMOT algorithm). Note the first two digits of the square root.

In the case of RSA-100, the first digit is three, and the second digit is nine. Treat this as 3.9. Now, the value of the square root is close to four. From the factors of 15, select the factors above one and below four. Consider this as the first digit of p (p1). Now, take the highest factor from the 15 factors that are situated above four. This is five (q1). Multiply p1 by q1 and compare with the first two digits of N. As the number is exactly equal to the first two digits of N, the highest factor obtained (q1) is reduced by one. This is to account for the fact that N as a whole number has more than two digits. If p1 and q1 were left as they are with three and five, the first digit and additional digits were added, the result would be a number that has a higher which does not have 15 as the first two digits. The highest factor q1 minus one is four. Now, p1 equals three, and q1 equals four. The factors of N are likely to start with three and four. Two variables are then created to accommodate two known factors of N. In the procedure, these are shown as variables p and q. As the two known factors of N are one and N itself, these are assigned to p and q. p and q are then adjusted by a process of multiplication and division until the numbers output match the first digits of p1 and q1. The log of N is then calculated, and through reverse addition, the next digits of p and q are obtained. Finding the inverse log helps to ensure the right digits are selected.

6 Evaluation

6.1 Main findings of the study

6.1.1 The Decimal method for achieving results for RSA-100

Note: The full process for obtaining these results is in the associating ICT solution (DERMOT algorithm) on page 28 of the accompanying configuration manual.

Results for prime factor p:

```
>>> d = Decimal('10') ** Decimal('0.579501')
>>> d
Decimal('3.797528133834932147373806559338114390765747988496745096816587305693150844698974285641424726880298177 04')
```

The actual prime factors are

RSA-100

p = 3797522793694367392280887275544562785456553663819 9

q = 4009469095092088103068373529276146838921489972406 1

The results show that the first six digits of p were successfully matched. If the digits after the decimal place are ignored, the last digit in q is 4. The lengths of p and q are approximately half of N. As the last digit is four, there is no need to do a primality test. The last digit of N is 9, which means the last digit of q should be 1, 3, or 9.

6.2 Implications of Findings

6.2.1 Business perspective

A ransomware attack in the operational technology (O.T.) industry can have devastating consequences. Such an attack may involve an attack on information technology (I.T.) systems controlling the O.T. or an attack directly on the O.T. Obtaining a decryption key swiftly is of paramount importance. As well as the risks that impact the I.T. industry, the O.T. industry has additional risks. For example, systems are often not updated because updating software or rebooting systems can affect processes. Human safety is the main concern in the operational technology industry. If a control system exceeds a specified value, a safety instrumented system that communicates with the control system brings the system to a safe state. Any interference with the threshold level could have varying levels of consequences. For example, a change in the level of a chemical system could result in severe burns or the death of a person. In the event of a ransomware attack, decryption keys are not always available, as new strains often emerge. In this situation, a company in the operational technology industry needs a time-focussed solution for finding a decryptor key. Once a ransomware name has been discovered, the incident response team can use publicly available decryption tools such as nomoreransom.org[6] to obtain the decryption key and instructions on how to use it appropriately. However, often, a decryption key is not available as new strains of ransomware emerge.

Use Case 1: Application in the Operational Technology (O.T.) industry.

A pharmaceutical manufacturing company is performing a risk assessment. As part of the risk assessment, the security of its encrypted devices is tested.

[6]https://www.nomoreransom.org/en/index.html

The company needs a solution for obtaining a decryption key in the event of a ransomware attack. The solution needs to be adaptable for Windows and LINUX systems. An attack could result in the wrong quantity of chemicals in tablets, which could have dangerous implications for human health. It could also result in a chemical explosion, which could result in burns or death. The ideal solution can be used anywhere. Because the DERMOT algorithm is implemented in Python it can easily be used anywhere. The incident response team can copy the encrypted file and store it on an isolated Windows machine for analysis.

RSA is often used in combination with a symmetric key algorithm such as AES. The RSA algorithm is often used for key exchange, and AES is used for data encryption. If an RSA algorithm has been used in conjunction with AES encryption, the solution presented here (the DERMOT algorithm) , once fully established, can be applied. First, search the encrypted file for the AES password. Run the algorithm. The output produced is the decryptor key. Use this key to decrypt the AES password into plaintext.

6.2.2 Academic perspective

This algorithm can be used as an example to students in disciplines such as mathematics, chemistry, astrophysics, and computers of how to quickly eliminate most numbers between one and N. This can be applied where significantly large numbers are used, such as in astrophysics.

Discussion

The algorithm is based on the factorisation of semi-prime numbers using decimals. When refined, it will lead to factorisation of semi-primes with a predictable increase in N. The way to achieve this is to assign its known factors to two variables representing p and q. The known factors are one and the semi-prime number itself.

The key to attaining a result close to the desired outcome is to obtain the factors of the first two digits of N. In most cases, the highest factor obtained, excluding one and N itself, should be reduced by one. However, this is just an initial estimation. Another limitation of this work is that it is based on a very small sample size of just sixteen semi-prime numbers.

The square root is set as the maximum threshold for one prime factor and the minimum threshold for the other prime factor. Assuming most prime numbers do lie close to the square root, this suggests that the generation of semi-prime numbers is not random. Alternatively, it may be that a specified range is pre-determined to enhance security. If the semi-primes were farther away from the square root, then a computer that is running a program to factorise the semi-prime would achieve factorisation much faster. From a security perspective, this is not a positive outcome if a malicious person is attempting to decrypt confidential data.

Having assigned the variables p and q with initial values of one and the semi-prime number itself, p and q are then adjusted through a process of multiplication on one variable and division on the other variable. Using logs is helpful, but the number of steps required to match a digit can make the time taken to obtain prime factors become unpredictable. This is a limitation of this work and identifies the need for further investigation. The reason this is an issue is because when trying to decide how the

digits in p and q add up to provide a given digit in the log of N, there are a few different possibilities. The most suitable arrangement of digits may be achieved on the first attempt or maybe after two, three, or four attempts. It also depends on the digit in the log of N. For example: If the log of N contains digit one, then the possibilities are (p=0 and q=1), (p=1, q=0) (p=2 and q=9), (p=9 and q=2), (p=3 and q=8), (p=8 and q=3), (p=4 and q=7), (p=7 and q=4), (p=5 and q=6), (p=6 and q=5). Additionally, if there is one carried over, this means that more combinations are possible. An important aspect then of using logs is to carefully consider which number to select for log p and log q. Obtaining the inverse log of possible combinations can help with this process.

Having assessed the process, it could be argued that both Flynn (2021) and Bellos made valid points. Flynn (2021) claimed too many iterations were required using the log method. This is true if testing all odd numbers between one and N. However, the DERMOT algorithm reduces the number of semi-prime numbers to be tested by first establishing the first six digits. Then, all digits not matching the first six digits can be automatically eliminated. As the RSA semi-primes are extremely large numbers this is a significant portion of numbers that are excluded from testing. As Bellos (2010) stated, the reason Johannes Kepler was able to calculate the orbit of Mars was due to logs.

Conclusion and future work

The focus of this work was to establish how to speed up the process of recovering a decryption key in the event of a ransomware attack that employs the use of the RSA algorithm. The RSA algorithm is used in the exchanging of key processes and is often employed with an encryption scheme such as AES. To decrypt the AES the RSA decryption key is needed. This key is generated through a process that begins with multiplying two prime numbers together to produce a semi-prime number. To speed up the process for obtaining the decryption key an alternative approach to the approaches already attempted is required. Because the focus is on obtaining the decryption key as quickly as possible, the calculation is based on the decimal system and uses logs and inverse logs to refine results.

There were three main objectives. Objective one was to create a unique algorithm to reduce the number of computations required for the factorisation of a semi-prime number without the use of a quantum computer. The 100-digit number that was attempted was not factorised into its prime factors. However, the first six digits were discovered, and the process was applied to small numbers with 100% accuracy. With more refinement, this can be achieved without the use of a quantum computer. The second objective was to implement the algorithm with suitable software. The DERMOT algorithm was implemented with Python, which was 100% suitable for the task. The third objective was to provide use cases for the O.T. industry. The main consideration for the O.T. industry is that I.T. is connected to O.T. which are often essential services and could have devastating consequences if infected with ransomware. Any software updates need to consider the potential implications of connected O.T. For that reason, it is recommended to copy the encrypted file and store it on an isolated Windows machine for analysis. If an RSA algorithm has been used in conjunction with AES encryption, the DERMOT solution presented here can be used to factorise the semi-prime number and the process of recovering the decryption key can take place.

This algorithm is undergoing further testing using other semi-prime numbers and the DERMOT algorithm is being automated. Other future work could include investigating the role artificial intelligence (A.I.) could play in this process. Specifically, an A.I. model trained to identify the next digit

of log p and log q based on the digits in the log of N would be helpful. Relevant contributions could include an updated survey on O.T. systems impacted by ransomware.

Acknowledgements:

I would like to thank my supervisor Ross Spelman for his guidance with this research. I would also like to thank Gina Quin and the 30% club for providing me with a scholarship to continue my studies and complete the MSc. in Cybersecurity. I would also like to thank many others who have contributed to my education including my parents and Springboard and all the lecturers at the National College of Ireland who made learning an interesting and enjoyable experience. This paper is dedicated to my Uncle, who passed away during this research.

References

Pech. R (2006), *NewScientist*, Issue No. 2578, Melbourne, Australia

Easttom, W. (2021). *Modern cryptography: Applied mathematics for encryption and information security.* Cham, Switzerland. Springer.

Park, H.B., Sim, B.Y., and Dong-Guk Han, D.G. (2021). 'Machine Learning-Based Profiling Attack Method in RSA Prime Multiplication.' In *Proceedings of the 2020 ACM International Conference on Intelligent Computing and its Emerging Applications (ACM ICEA '20)*. Association for Computing Machinery, New York, NY, USA, December 2020, Article 32, 1–6. https://doi.org/10.1145/3440943.3444730

Katz, J., & Lindell, Y. (2020). *Introduction to modern cryptography* (3rd ed.). Boca Raton, Florida, United States. Chapman and Hall/CRC.

Wang, X. (2022). 'Progress in Applying Valuated Binary Tree to Factorize Big Integers.' In *Proceedings of the 2022 7th International Conference on Intelligent Information Technology (ICIIT '22)*. Association for Computing Machinery, New York, NY, USA, February 2022, pp. 90–94. https://doi.org/10.1145/3524889.3524904

Sabani, M., Galanis, I., Savvas I., and Garani, G. (2022). 'Implementation of Shor's Algorithm and Reliability of Quantum Computing Devices.' In *Proceedings of the 25th Pan-Hellenic Conference on Informatics (PCI '21)*. Association for Computing Machinery, New York, NY, USA, November 2021, pp. 392–396. https://doi.org/10.1145/3503823.3503895

Nemec, M., Sys, M., Svenda, P., Klinec, D., and Matyas, V. (2017). 'The Return of Coppersmith's Attack: Practical Factorisation of Widely Used RSA Moduli.' In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security (CCS '17)*. Association for Computing Machinery, New York, NY, USA, October 2017, pp.1631–1648. https://doi.org/10.1145/3133956.3133969

Joshi, S., Bairwa, A.K., Pljonkin, A.P., Garg, P., and Agrawal, K. (2023). 'From Pre-Quantum to Post-Quantum RSA.' In *Proceedings of the 6th International Conference on Networking, Intelligent Systems & Security (NISS '23)*. Association for Computing Machinery, New York, NY, USA, May 2023, Article 1, 1–8. https://doi.org/10.1145/3607720.3607721

IBM. (2024). *IBM X-Force Threat Intelligence Index 2024*.
Available at: https://www.ibm.com/reports/threat-intelligence [Accessed 01 May 2024]

MSSP Research Lab (2024) *Malware Analysis Report: Blackcat ransomware.* Available at: https://mssplab.github.io/threat-hunting/2023/07/13/malware-analysis-blackcat.html [Accessed 01 May 2024]

SentinelOne (2024) *Clop.* Available at: https://www.sentinelone.com/anthology/clop/

[Accessed 01 May 2024]

Stephen, C. (2023) *What is RSA-4096 Ransomware & How to Protect Against It?*
Available at: https://www.comparitech.com/net-admin/rsa-4096-ransomware/
[Accessed 15 April 2024]

Elsad, A. (2022) *Threat Assessment: Black Basta Ransomware*. Available at: https://unit42.paloaltonetworks.com/threat-assessment-black-basta-ransomware/
[Accessed 01 May 2024]

Emsisoft (2021) *Let's talk about Conti.*
Available at: https://www.emsisoft.com/en/blog/37866/ransomware-profile-conti/
[Accessed 15 April 2024]

Cyware (2019) *Let's Talk About LockBit - An In-depth Analysis.*
Available at: https://cyware.com/resources/research-and-analysis/lets-talk-about-lockbit-an-in-depth-analysis-7cf0
[Accessed 15 April 2024]

Santos, D., Bunce, D., Galiette, A. (2023) *Threat Assessment: Royal Ransomware*. Available at: https://unit42.paloaltonetworks.com/royal-ransomware/ [Accessed 01 May 2024]

RSA Factoring Challenge Administrator (challenge-administrator@majordomo.rsasecurity.com) (January 30, 2002) [March 5, 1999]. "RSA Honor Roll". challenge-rsa-honor-roll@rsa.com (Mailing list). Archived from the original on September 9, 2023 – via Ray Ontko.

Overmars, A. and Venkatraman, S. (2021). 'New semi-prime factorisation and application in large RSA key attacks.' *Journal of Cybersecurity and Privacy*. Vol.1. pp. 660-674. 10.3390/jcp1040033

Inan, A. (2022) 'Method for Approximating RSA Prime Factors'. *In Proceedings of the 2022 European Interdisciplinary Cybersecurity Conference (EICC '22).* Association for Computing Machinery, New York, NY, USA, 1–5. https://doi.org/10.1145/3528580.3528581

Rodriguez G., R., Castang M., G., Vanegas, C.A. (2021). 'Information Encryption and Decryption Analysis, Vulnerabilities and Reliability Implementing the RSA Algorithm in Python'. In: *Florez, H., Pollo-Cattaneo, M.F. (eds) Applied Informatics. ICAI 2021. Communications in Computer and Information Science*, vol 1455. Springer, Cham. https://doi.org/10.1007/978-3-030-89654-6_28 [Accessed 07 July 2024]

Flynn, S. (2021) 'An Investigation into the Mathematics of Decryption Techniques in RSA Encryption with an Implementation in Python' *Georgetown Scientific Research Journal,* Vol. 1 Ed.2 pp. 7-15.

Bellos, A. (2020) *Alex's Adventures in Numberland.* 10th Edition. London. Bloomsbury Publishing.

RSA Digital Risk Management & Cyber Security Solutions (2019). *RSA Digital Risk Management & Cyber Security Solutions*. [online] RSA.com. Available at: https://www.rsa.com/.

Python (2019). *Download Python*. [online] Python.org. Available at: https://www.python.org/downloads/.

Wolfram Alpha (2024). *Wolfram|Alpha: Making the world's knowledge computable*. [online] Wolframalpha.com. Available at: https://www.wolframalpha.com/.

sagecell.sagemath.org. (n.d.). Sage Cell Server. [online] Available at: https://sagecell.sagemath.org/.