REPUBLIC OF TÜRKİYE

AKDENIZ UNIVERSITY

INVESTIGATING ATTACKS ON RSA CRYPTOGRAPHIC ALGORITHM WITH MACHINE LEARNING TECHNIQUES

Ahmet GÜRDAL

INSTITUTE OF NATURAL AND APPLIED SCIENCES

DEPARTMENT OF COMPUTER ENGINEERING

MASTER THESIS

OCTOBER 2024

ANTALYA

REPUBLIC OF TÜRKİYE

AKDENIZ UNIVERSITY

Investigating Attacks on RSA Cryptographic Algorithm With Machine Learning Techniques

Ahmet GÜRDAL

INSTITUTE OF NATURAL AND APPLIED SCIENCES

DEPARTMENT OF COMPUTER ENGINEERING

MASTER THESIS

OCTOBER 2024

ANTALYA

REPUBLIC OF TÜRKİYE

AKDENIZ UNIVERSITY

INSTITUTE OF NATURAL AND APPLIED SCIENCES

Investigating Attacks on RSA Cryptographic Algorithm With Machine Learning Techniques

Ahmet GÜRDAL

DEPARTMENT OF COMPUTER ENGINEERING

MASTER THESIS

This thesis was unanimously accepted by the jury on 14/10/2024.

Asst. Prof. Dr. Murat AK (Advisor)

Assoc. Prof. Dr. Taner DANIŞMAN

Asst. Prof. Dr. Naci ER

ABSTRACT

Investigating Attacks on RSA Cryptographic Algorithm With Machine Learning Techniques

Ahmet GÜRDAL

MSc Thesis in Computer Engineering

Supervisor: Asst. Prof. Dr. Murat AK

October 2024, 74 pages

This thesis examined the application of machine learning models for factorizing semi-prime numbers to address the RSA cryptanalysis problem. Across three bit-lengths, ten datasets were trained on seven different neural network topologies, resulting in a total of 165 models. Despite extensive efforts, the models struggled to predict the outputs accurately, highlighting the inherent difficulty of factorizing semi-prime numbers and the robust security of RSA encryption. An analysis of bit frequency distributions revealed no discernible patterns, further demonstrating the complexity of prime factor distributions in semi-primes.

Experiments with low-bit-length datasets indicated that increasing the dataset size and training epochs could improve prediction accuracy. Additionally, as bit lengths grow, the need for custom-designed neural networks tailored to this specific problem becomes crucial to optimize both accuracy and training efficiency. Although the results did not meet initial expectations, they provide valuable insights into cryptanalysis methodologies. The findings emphasize the importance of refining machine learning techniques, expanding datasets, and adopting interdisciplinary approaches to advance cryptanalysis and strengthen digital communication security.

KEYWORDS: Binary Approach, Cryptanalysis, Encryption, Machine Learning, Neural Network

COMMITTEE: Asst. Prof. Dr. Murat AK

Assoc. Prof. Dr. Taner DANIŞMAN

Asst. Prof. Dr. Naci ER

ÖZET

RSA Kriptografik Algoritma Saldırılarının Makine Öğrenimi Teknikleri ile Araştırılması

Ahmet GÜRDAL

Yüksek Lisans Tezi, Bilgisayar Mühendisliği Anabilim Dalı
Danışman: Dr. Öğr. Üyesi Murat AK

Ekim 2024, 74 Sayfa

Bu tez, yarı-asal sayıların çarpanlara ayrılması ve RSA kriptanalizine yönelik makine öğrenimi modellerinin uygulanmasını incelemiştir. Üç farklı bit uzunluğu boyunca, on veri seti yedi farklı sinir ağı topolojisiyle eğitilmiş ve toplamda 165 model oluşturulmuştur. Yoğun çalışmalara rağmen modellerin, çıktıları doğru bir şekilde tahmin edememesi, çarpanlara ayırma probleminin zorluğunu ve RSA şifrelemesinin güvenliğini ortaya koymuştur. Bit frekansı dağılımlarına yönelik analizler ise belirgin bir desen ortaya koymamış ve çarpanların sonuç içerisindeki dağılımlarının karmaşıklığını göstermiştir.

Düşük bit uzunluğundaki verilerle yapılan deneyler, veri seti boyutunun artırılması ve eğitim dönemlerinin (epoch) uzatılmasının tahmin doğruluğunu artırabileceğini göstermiştir. Ayrıca, bit uzunluğu arttıkça, bu özel probleme uygun şekilde tasarlanmış özelleştirilmiş sinir ağı modellerinin kullanılması hem doğruluk hem de eğitim verimliliği açısından kritik öneme sahiptir. Sonuçlar başlangıçtaki beklentileri karşılamasa da kriptanaliz yöntemlerine ilişkin önemli bulgular sunmaktadır. Bu çalışma, makine öğrenimi tekniklerinin geliştirilmesi, veri setlerinin genişletilmesi ve disiplinler arası yaklaşımların benimsenmesinin kriptanaliz alanında ilerleme sağlamak ve dijital iletişim güvenliğini güçlendirmek için gerekli olduğunu vurgulamaktadır.

ANAHTAR KELİMELER: İkili Yaklaşım, Kriptoanaliz, Makine Öğrenimi, Sinir Ağı, Şifreleme

JÜRİ: Dr. Öğr. Üyesi Murat AK
Doç. Dr. Taner DANIŞMAN
Dr. Öğr. Üyesi Naci ER

ACKNOWLEDGEMENTS

First and foremost, I want to convey my appreciation to my advisor, Murat AK for their assistance with the writing of this thesis.

Despite being far away, my friends Salih K., Yunus A. and Vedat Y. have provided invaluable support through our discussions on the thesis study and I want to thank them for their suggestions on the thesis.

Finally, in the next line, I wish to extend my heartfelt thanks to my beloved family in Turkish for their unwavering moral support while I wrote this thesis.

Sevgili ailem, karşılaştığımız tüm zorluklara rağmen beni hep desteklediğiniz ve her zaman yanımda olduğunuz için çok teşekkür ederim.

CONTENTS

TEXT OF OATH

I declare that this study "INVESTIGATING ATTACKS ON RSA CRYPTOGRAPHIC ALGORITHM WITH MACHINE LEARNING TECHNIQUES", which I present as a master thesis, is by the academic rules and ethical conduct. I also declare that I cited and referenced all material and results that are not original to this work.

14/10/2024

Ahmet GÜRDAL

SYMBOLS AND ABBREVIATIONS

Symbols:

$\mathbb{Z}_0^-$: $\mathbb{Z}^- \cup \{0\}$

$\mathbb{N}_0$: $\mathbb{N} \cup \{0\}$

Abbreviations:

AFE	: Autoencoder Feature Extraction
AI	: Artificial Intelligence
ANN	: Artifical Neural Network
BP	: Back Propagation
BPVS	: Back Propagation with Variable Stepsize
CFF	: Convolutional Feature Fusion
CSV	: Comma Separated Value
DNN	: Deep Neural Network
FIN	: Feature Interaction Network
FP	: Feed Forward Propagation
GA	: Genetic Algorithm
GMP	: GNU Multiple Precision Arithmetic Library
HCL	: Hybrid Convolutional LSTM
HN	: Hierarchical Network
LSTM	: Long Short-Term Memory
MD	: Multi Dense
NN	: Neural Network
OABP	: On-Line Adaptive Back Propagation
RNN	: Recurrent Neural Network
RPROP	: Resilient Back Propagation
SNN	: Simulated Neural Network

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

From the inception of the Internet, secure data transfer has been a critical concern in the field of computer security. Even before the Internet, various methods were employed to securely store or transmit messages, such as Ancient Spartan Cryptography, Roman encryption and ciphers, the Hebern rotor machine, and World War II cryptography. Throughout history, we have relied on encryption methods, algorithms, and protocols to meet our need for secure communication. Today, mathematicians and programmers continue to develop new encryption techniques to ensure that only the intended recipients can access transmitted messages. Numerous functions have been created to encrypt data, some relying on secret keys and others utilizing mathematical formulas.

The primary aim of this thesis is to investigate attacks on the RSA factoring problem using machine learning techniques and to seek improved methods for compromising the RSA cryptosystem. By researching and testing current machine learning approaches to attack the encryption algorithm, we aim to gain a deeper understanding of their advantages and weaknesses. Additionally, by comparing historical research results with newly generated data, we can visualize the actual improvements that have been made. The analysis methods employed in this thesis will consist of machine learning models and mathematical operations. These methods will be designed to produce meaningful data that can be compared with other encryption algorithms based on performance, security, and reliability.

In conclusion, the thesis will compare and summarize the definitions of the RSA encryption algorithm, its data processing methods, and the outputs obtained from the analysis. Given the rapid advancements in machine learning algorithms, it is possible that, with the help of these models, we can identify and analyze any hidden patterns between cipher texts and plain texts, should they exist.

1.1. Cryptography

The process of encrypting communications and data with codes so that only the intended recipient can decode them and thereby preventing unauthorized access, is known as cryptography. The term "cryptography" derives from the Greek words "crypt," meaning

"hidden," and "graphy," meaning "writing." Data security techniques in cryptography are based on mathematical ideas and a set of calculations controlled by rules, or algorithms, that change signals into forms that are challenging to understand. These algorithms are employed to verify data privacy, digitally sign documents, generate cryptographic keys, facilitate secure internet browsing, and protect private transactions such as credit card purchases.

In general, there are two types of cryptography:

- ***Symmetric Key Cryptography:*** In this type of encryption, both the sender and the receiver use the same key to encode and decode messages. Although these systems are typically faster and simpler to implement, they require a secure method for exchanging the key beforehand, which can be a significant drawback. The two most popular symmetric key encryption methods are the Data Encryption Standard (DES) and the Advanced Encryption Standard (AES).

- ***Asymmetric Key Cryptography:*** This encryption method uses a pair of keys (named public and private keys) for encrypting and decrypting data. The public key of the recipient is used for encryption, while the private key is used for decryption. These two keys are different; only the recipient has access to their private key. This ensures that even if the public key is publicly available, only the recipient can decrypt the message. The most popular technique for asymmetric key encryption is the RSA algorithm.

1.1.1. RSA Operation and Prime Numbers

The asymmetric key cryptosystem called RSA allows digital signatures and secure communication over unreliable networks. In 1977, it was Created by cryptographers and computer scientists, Ron Rivest, Adi Shamir, and Leonard Adleman.

This is how the RSA algorithm works:

1. **Key Generation**

    - **Choosing Two Large Prime Numbers**; for security purposes, choosing big prime numbers is essential. The challenge of factoring the products of these primes is what gives RSA its security.

- **Semi-prime**; calculate the modulus N as the product of the chosen primes as $N = pq$.
- **Totient**; calculate the Euler's totient function $\phi(N)$ which is the count of positive integers less than N that are coprime with N. And it equals to $\phi(N) = (p-1)(q-1)$
- **Choose Public Exponent(e)**; Select a public exponent e that is coprime to $\phi(N)$. 65537, which is a prime number, is the most common choice for the value of e.
- **Calculate Private Exponent(d)**; Determine the private exponent d with using this formula; $d * e \equiv 1 \mod \phi(N)$. So, it is necessary to find the modular multiplicative inverse of e modulo $\phi(N)$.

2. **Key Distribution**

- **Public Key**; (N,e) is published and can be seen by everyone.
- **Private Key**; (N,d) should be hidden by the owner and only the owner should have access to it.

3. **Encryption**

- **Message Representation**; is the method of representing the plaintext message as an integer m such that $0 \leq m < N$
- **Cipher Calculation**; compute the ciphertext c using the public key with this formula $c \equiv m^e \mod N$. This modular exponentiation operation ensures that the result is within the modulus N.
- **Sending Ciphertext**; transmit the ciphertext c to the intended recipient using any suitable transfer method.

4. **Decryption**

   - **Cipher Reception**; the recipient, possessing the private key, receives the ciphertext c.
   - **Decryption Calculation**; decrypting the ciphertext c using the private key with the formula $m \equiv c^d \mod N$.
   - **Reverse Message Representation**; since the value calculated with the decryption formula is the same as the value encrypted by the sender. This value m can be changed back to the plaintext version by using the same converter method used on the sender side but reversed.

The difficulty of factoring the product of two large primes which is believed to be a challenging mathematical problem, is the foundation of the security of RSA. As computational power increases, key lengths must be adjusted to maintain the same level of security. RSA is widely utilized in digital signatures, key exchange systems, and secure communications due to its robust security and adaptability, making it an indispensable algorithm in modern cryptography.

1.2. Training Theory

Every minute, technology becomes more deeply integrated into various aspects of our daily lives. To meet the growing expectations of consumers, businesses are increasingly relying on machine learning algorithms to streamline their operations. This reliance is particularly evident in social media, where machine learning is used for tasks such as object detection in images, and in the realm of personal electronics, where virtual assistants like Alexa and Siri utilize machine learning for direct communication.

While machine learning (ML) and Neural Networks (NN) are related technological terms commonly used in this project, it is necessary to cover the basis of these terms and their differences before defining the aim of the project.

1.2.1. Machine Learning

Machine learning is a subfield of artificial intelligence (AI), which is the study of intelligent human behavior. Artificial intelligence systems can now complete complicated

jobs in a way that is similar to how humans solve problems.

In the 1950s, Arthur Samuel of IBM developed computer software for playing checkers, a significant early milestone in the field of artificial intelligence. Samuel's software was constrained by limited computer memory, prompting the development of techniques like alpha-beta pruning. Central to his approach was a scoring system based on the positions of pieces on the board, aiming to estimate each player's likelihood of winning. This scoring function formed the basis of the minimax strategy, which ultimately led to the development of the minimax algorithm. This algorithm enabled the software to decide its next move by evaluating potential outcomes and selecting the one that maximized its chances of success.

Samuel additionally created a variety of mechanisms that improved his program. Samuel referred to this process as rote learning, in which his program coupled the values of the reward function with a record of every position it had previously encountered. In 1952, Arthur Samuel first used the term "machine learning." So, the pioneer of AI, Arthur Samuel originally described machine learning as the area of research that enables computers to learn without being explicitly instructed.

Data; such as texts, numbers, and even images; are the fuel of the machine learning models. Examples of data include time series data from sensors, bank transactions, images of objects or individuals, sales reports, etc. Before being employed as training data, which constitutes the information used to train the machine learning model, data must be gathered and prepped. The result of the learning process is better when more data is collected.

The neural network model is trained to identify patterns and make predictions by programmers who select a machine-learning model, provide data, and monitor its performance. To enhance the accuracy of the model, human programmers may adjust its parameters and make other modifications over time. This iterative process allows for continual improvement of the performance of the model and the generation of more accurate results.

A subset of the training data is masked and employed as evaluation data to confirm that the machine learning model is accurate when fresh data is introduced. The final version of the model that can be used for different data sets in the future will be the end product

of the training process.

Machine learning systems can act like different tools depending on their purpose. There are three main ways these systems can be used:

- **Descriptive:** Imagine a machine learning system like a detective looking at clues (data). A descriptive system would analyze the data and tell you what happened in the past. For example, a system analyzing sales data could tell you which products sold the most last month.

- **Predictive:** Now, imagine the detective is trying to solve a future crime (predict what will happen). A predictive system would use the data to make educated guesses about what might occur in the future. Building on the sales example, the system could predict which products are likely to be popular next month based on past trends.

- **Prescriptive:** Think of the detective not just predicting a crime, but also suggesting how to prevent it. A prescriptive system would analyze the data and recommend actions to take. In our sales example, the system might suggest stocking up on certain products or running promotions based on its predictions.

The type of machine learning system you choose depends on what you want to achieve. Essentially, the selected machine learning system can make us understand what happened in the past (descriptive), predict what will happen in the future (predictive), or get recommendations on what actions to take (prescriptive).

After the training of a neural network model, two critical challenges can hinder the effectiveness of the model: overfitting and underfitting.

- **Overfitting:** This occurs when a model memorizes the training data too precisely, capturing even the noise and idiosyncrasies within that data. While the model performs exceptionally well on the training data itself, it generalizes poorly to unseen data. Imagine a student studying only test questions and replicating them perfectly on the exam; they wouldn't understand the underlying concepts and struggle with new problems.

- **Underfitting:** Underfitting occurs when a model is too basic and misses important patterns in the training data. This leads to low performance on both new data and training data. It is similar to students who haven't studied enough – they don't have the knowledge to answer questions well.

Both overfitting and underfitting prevent the model from achieving its goal of generalizability – the ability to make accurate predictions on new data. A well-trained model strikes a balance between learning the underlying patterns and avoiding memorization of specifics.

To create a successful machine learning algorithm, first, which type of machine learning technique to be used must be decided. There are four types of machine learning;

- ***Supervised Learning:*** Supervised learning uses labeled datasets to train models for accurate outcome prediction or data classification. The model adjusts its parameters when new data is inputted until it achieves a good fit. This adjustment occurs during cross-validation to prevent overfitting or underfitting. Organizing spam emails into a separate folder is just one example of how large organizations utilize supervised learning to tackle real-world problems. Techniques employed in supervised learning include neural networks, Naïve Bayes, logistic regression, random forest, linear regression, and support vector machines (SVM).

- ***Unsupervised Learning:*** Unsupervised learning involves using machine learning algorithms to analyze and group datasets that don't have labels. These algorithms discover hidden relationships or patterns in the data without human intervention. Unsupervised learning is useful for tasks like segmenting consumers, suggesting complementary products, exploring data, and recognizing patterns and images because it can identify similarities and patterns in the data. It can also help reduce the number of features in a model through dimensionality reduction. Popular methods for this include principal component analysis (PCA) and singular value decomposition (SVD). Neural networks, probabilistic clustering approaches, and k-means clustering are among other algorithms utilized in unsupervised learning.

- ***Semi-supervised learning:*** Semi-supervised learning provides a satisfying middle ground between supervised and unsupervised learning. It employs a smaller labeled

data set for training purposes, which directs the classification and feature extraction processes from a larger unlabeled data set. When there is insufficient labeled data for a supervised learning system, semi-supervised learning can help. When labeling sufficient data is too expensive, it also helps.

- ***Reinforcement learning***; is a computational approach to learning where an agent interacts with an environment to achieve a goal. Through trial and error, the agent learns to take actions that maximize a cumulative reward signal. Every time step, the agent assesses the condition of the environment, chooses a course of action, and gets feedback in the form of a reward signal that indicates the immediate result of the action. The goal of an agent is to find a pattern or learn/understand a policy and it tries to achieve that goal by establishing a state-to-action mapping that optimizes the anticipated cumulative reward over time. Reinforcement learning algorithms employ various strategies, such as exploration and exploitation, to balance between discovering new actions and exploiting known ones. Reinforcement learning has applications in diverse fields, such as finance, robotics, business management, healthcare, etc. where autonomous decision-making in dynamic environments is crucial.

To determine which machine learning system, should be selected to apply in a project, there is also a decision map suggestion created by Thomas W. Malone and designed by Laura Wentzel (Brown 2021). As this map suggests if data that is provided must be clustered naturally, choosing the unsupervised learning system; or else, if the model must learn what actions to take in different situations then supervised or reinforcement learning would be more suitable.

If the learning process needs to be active, it implies that the decisions made by the system will impact the situations it encounters later on. In contrast, if the learning process is passive, the system learns solely from the given data without actively influencing subsequent situations.

1.2.2. Neural Network

Deep learning methods rely on neural networks, often called simulated neural networks (SNNs) or artificial neural networks (ANNs). Neural networks are a type of machine learning model inspired by the structure and functioning of the human brain. They mimic how neurons in the brain communicate with each other.

Neural Networks have been used to speed up and automatize various tasks. Such as;

- Medical diagnosis by image classification.
- Targeted marketing by social network filtering and behavioral data analysis.
- Financial predictions by analyzing the historical data of financial instruments.
- Electrical load and energy demand forecasting.
- Process and quality control.
- Chemical compound identification.
- Document analysis; automated virtual agents and chatbots by natural language processing.
- Recommendation Engines by user activity analysis.
- More accurately convert speech to text for meeting recordings and subtitles, assist call center agents and automatically sort calls using speech recognition.
- Self-driving cars, facial recognition, image labeling, etc. by using computer vision.

ANNs consist of one or more hidden layers, an output layer, and an input layer, which together form the node layers. Every artificial neuron, also called a node, in the network is linked to other nodes and possesses a weight and threshold. A node becomes active and sends data to the next layer of the network when its output surpasses a predetermined threshold value. On the other hand, no data is moved to the next layer of the network if the output falls short of the threshold.

Neural networks require training data in order to learn and as they are exposed to more data over time, their accuracy typically improves. Fortunately, after they are calibrated for accuracy, learning algorithms are effective tools in artificial intelligence and computer science that allow for quick data classification and clustering. Tasks requiring pattern recognition, which may have previously taken hours for manual detection by human experts, can now be accomplished in minutes with the assistance of neural networks. One of the most renowned examples of a neural network in practical use is the algorithm employed by Google for its search engine.

1.3. Aims and Objectives

1.3.1. Aim

The primary aim of the thesis is to assess previous methods used for decrypting cipher data. This involves conducting a comprehensive analysis of these methods, scientifically delineating their strengths and weaknesses, and identifying the underlying reasons for their limitations. Subsequently, the objective is to devise innovative strategies or algorithms that mitigate the shortcomings observed in prior approaches, with the ultimate goal of creating a decryption method that is more effective and successful.

Finally, by integrating the definitions and functionalities of these encryption algorithms with the analysis results, attempting to address the following questions:

- Which kind of attacks work better against the RSA encryption algorithm?
- Can machine learning algorithms be used to identify patterns in RSA encrypted messages that could reveal the prime factors used in the encryption?
- How effective are machine learning algorithms at breaking RSA encryption compared to traditional methods such as brute-force attacks or factoring algorithms?
- How can a machine learning model be used to attack the RSA encryption algorithm?
- What kind of machine learning methods have been used to attack the RSA encryption algorithm?
- Can we find out more effective methods?

- Can we use machine learning methods together with classical algorithms in a hybrid manner to get better results?

1.3.2. Objectives

The primary objective of this project is to develop an optimal machine-learning model capable of efficiently factorizing semi-prime numbers. The model will take a semi-prime number $(n = pq)$ as input and aim to output either:

- The two prime factors of the semi-prime number, (p and q)
- One of the prime factors, (p or q)
- The totient of the semi-prime number (since determining the totient also allows for the identification of the prime factors). $(\phi(N) = (p-1)(q-1))$

By achieving this, the project seeks to enhance current methodologies in cryptanalysis, providing a powerful tool for breaking the security of encryption algorithms that rely on the difficulty of factorizing large semi-prime numbers. This advancement is expected to have significant implications for the field of cryptography. Firstly, it will contribute to a deeper understanding of the vulnerabilities present in widely-used cryptographic systems such as RSA, which is fundamental to many secure communications. Secondly, the project aims to push the boundaries of machine learning applications in cryptanalysis, demonstrating the potential of these advanced techniques to solve complex mathematical problems that have traditionally been resistant to computational attacks.

Furthermore, by integrating innovative machine learning strategies, the project aims to inspire future research and development in the field, encouraging the exploration of novel approaches to cryptographic challenges. This could lead to the discovery of new algorithms and techniques that enhance the overall security and reliability of digital communication systems. Overall, this project represents a significant step forward in the ongoing effort to secure data in an increasingly digital world, ensuring that sensitive information remains protected against evolving threats.

1.4. Overview of Thesis

This thesis explores the intersection of machine learning and cryptanalysis, focusing on the challenge of factorizing semi-prime numbers, which underpins the security of many cryptographic systems. The primary objective is to develop an advanced machine-learning model capable of efficiently breaking down semi-prime numbers into their constituent prime factors or calculating their totient, thereby providing insights into the strengths and vulnerabilities of current encryption algorithms.

The thesis is structured as follows:

1. **Introduction:**
    - A comprehensive overview of cryptographic systems and the importance of encryption in securing data.
    - An explanation of the RSA algorithm and the significance of the semi-prime factorization problem.
    - The motivation and objectives of the thesis.
2. **Literature Review:**
    - Examinations of historical and contemporary methods used in cryptanalysis, focusing on those targeting the RSA algorithm.
    - Reviews of machine learning techniques previously applied to similar problems.
    - Identification of gaps and limitations in existing research.
3. **Material & Method:**
    - Detailed descriptions of the machine-learning models considered for this project.
    - The dataset preparation, including the generation and selection of semi-prime numbers.
    - The process of training, validating, and testing the models.
    - Criteria for evaluating model performance.

4. **Results and Discussion:**

    - Presentation of the experimental results, including the performance metrics of different models.
    - A scientific delineation of the strengths and weaknesses of each approach.

5. **Conclusion:**

    - Summary of the key findings and their significance.
    - Recommendations for future research directions.
    - Potential improvements and applications of the developed machine-learning model.

This thesis aims to contribute to the field of cryptanalysis by leveraging machine learning to address the complex problem of semi-prime factorization, offering new perspectives and solutions for enhancing the security of cryptographic systems.

2. LITERATURE REVIEW

Since the implementation of the RSA cryptography system, numerous research experiments have been conducted and published regarding the RSA cryptography algorithm and its vulnerabilities. Given that the strength of the RSA algorithm relies on the complexity of the prime factorization problem, these studies typically focus on identifying weaknesses or patterns that could potentially undermine this complexity. This chapter includes reviews of several published research papers on the subject.

2.1. Difficulty of Breaking RSA

There are several studies that have looked at the connection between the difficulty of integer factorization and breaking the RSA algorithm, including *Breaking RSA Generically is Equivalent to Factoring* (Aggarwal and Maurer 2009) and *Breaking RSA May Be As Difficult As Factoring* (Brown 2016). These studies argue that the difficulty of breaking the RSA algorithm, specifically in obtaining the private exponent (d), is essentially the same as the difficulty of modulus(n) factoring into its prime components.

The core of the security of the RSA algorithm lies in the fact that while it is easy to multiply two large prime numbers to create a large composite number, it is challenging to reverse this process and factor the composite number back into its prime components. This factorization problem makes deriving the private key (d) from the public key (e, n) so challenging.

Research has shown that any efficient method for breaking RSA and recovering the private exponent would also provide an efficient solution for factoring large integers. Therefore, the security of RSA encryption is directly tied to the computational difficulty of integer factorization, reinforcing the notion that the two problems are computationally equivalent in terms of their complexity.

These insights underscore the robustness of RSA encryption, as long as the factorization of large integers remains a computationally hard problem. This equivalence between breaking RSA and factoring ensures that advances in one area directly impact the security considerations of the other.

2.2. RSA Attacks

In the study titled *RSA Attacks* (Alrasheed and Fatima 2021), detailed explanations of the various attacking methods developed over the years are provided.

- **Factoring Large Integers**

  This is the most widely known and one of the most challenging attack methods. However, due to its complexity, it does not pose a significant threat to the RSA system. The current fastest factorization algorithm is the General Number Field Sieve, with a run-time of $((c + o(1))n^{1/3} \log n^{2/3})$

- **Blinding Attack (Elementary Attacks)**

  A blinding attack on RSA exploits its mathematical properties to trick the private key holder into decrypting a message without realizing it. The simplified breakdown of the attack steps is as follows:

  1. The attacker has an encrypted message but no private key.
  2. They multiply the ciphertext by a random number, creating a blinded version.
  3. The attacker sends this blinded ciphertext to the private key holder for decryption.
  4. The private key holder decrypts it, unaware it is a blinded message.
  5. The attacker then uses the random number to reverse the process, revealing the original message.

  Elementary Attacks are essentially based on the users, the blinding attack is one of the elementary attacks.

  The formulas for the attack steps are as follows;

  1. **Blinding the Message**

     $$m' = m \cdot r^e \mod n$$

     where:

     – m is the original message.

- r is a random blinding factor.
- e is the public exponent.
- n is the modulus.

2. **Decrypting the Blinded Message**

$$c' = (m')^d \mod n$$

where:

- c' is the ciphertext of the blinded message.
- d is the private exponent.

3. **Revealing the Decrypted Message**

$$m = c' \cdot r^{-1} \mod n$$

where:

- r^{-1} is the modular inverse of r.

- **Low Private Exponent**

This attack depends on the theorem of Wiener which is;

Let $N = pq$ and $q < p < 2q$. Let $d < 1/3N^{1/4}$.

Since (N, e) is known the value of the d can be found if the d is sufficiently small.

The problem with this attack type is that the value of d is usually selected to be a huge number. Therefore, this attack becomes impractical to use.

- **Coppersmith's Short Pad Attack (Low Public Exponent)**

The theorem of this attack can be expressed in terms of this attack strategy;

While (N, e) is the public key. N is an n-bits-long number. $m = |n/e^2|$. Let $M \in Z_N^*$ be a message with length at most $n - m$ bits long. Define $M_1 = 2^m M + r_1$ and $M_2 = 2^m M + r_2$, where r_1 and r_2 are different integers with $0 \leq r_1, r_2 < 2^m$. If the attacker has (N, e) and the encrypted versions (C_1, C_2) of the separated message (M_1, M_2) (r_1, r_2 are not known). the value of M, can be recovered.

- **Implementation Attacks**

  Other attacks focus on exploiting weaknesses in implementing RSA rather than attacking the RSA algorithm directly. For example, an implementation attack categorized under timing attacks was discovered in the P. Kocher's paper, *Timing attacks on implementations of Die-Hellman, RSA, DSS, and other systems* (Kocher 1996). This attack allows an attacker to determine the value of the private exponent (d) by measuring the exact time taken for the decryption process. The steps of this attack are outlined below.

  Formula;

  - Let $d = d_n d_{n-1} ... d_0$ (binary of d)
  - Set $z = M$ and $C = 1$. For $i = 0...n$ do;
    - (1) if $d_i = 1$ set $C = C * z \mod N$
    - (2) $z = z * z \mod N$
  - C at the end has the value $M^d \mod N$

2.3. A First Study of the Neural Network Approach in the RSA Cryptosystem

The results of the initial experiments conducted on the RSA cryptosystem using a neural network are published in this paper. (Meletiou et al. 2002)

In the beginning, well-known RSA attack types were listed at the time as:

1. Given $y \equiv x^e \mod N$, trying all possible keys $0 \leq d \leq \phi(N)$ to obtain $x \equiv y^d \mod N$. However since $\phi(N) \propto N$ it is impossible.

2. Finding $\phi(N)$ and computing d such that

$$d * e \equiv 1 \mod \phi(N)$$

   (Euclidean Algorithm)

3. Finding d directly and computing $x \equiv y^d \mod N$.

4. Factoring $N = pq$, deriving $\phi(N) = (p-1)(q-1)$ and calculating d as above.

For training the network, these training methods are used extensively. However, it did not make a promising difference in the results.

- Standard Back Propagation (BP)
- Back Propagation with Variable Stepsize (BPVS)
- Resilient Back Propagation (RPROP)
- On-Line Adaptive Back Propagation (OABP)

As it is described in the paper as well if the value of d, then $e * d - 1$ can be calculated and this is a multiple of $\phi(N)$. In Miller's paper (Miller and Rieman 1976), it has been demonstrated that N can be factored using a multiple of $\phi(N)$. So, finding the value of d is as hard as finding or calculating the value of $\phi(N)$ or as factoring N.

The neural network designed in this project receives N (pq) as input and $\phi(N)$ which is $(p-1)(q-1)$ as output.

2.3.1. Results

There are several results of different training models in the paper. To test the network performances, *complete measure* (μ_0) which shows the percentage of the exact match cases of predictions and targets, and *near measure* ($\mu_\pm$) which gives the percentage of targets and predictions which are close to each other with a given range. However, networks are not trained with large p and q values.

While using $N \leq 10001$, the network with 1-5-5-1 Layer Topology trained with 80000 epoch using 66% train data. And it returned the accuracy of 3%(with train data) and 5% (with test) for μ_0. For μ_5 these values increase to 35% and 40% respectively. And lastly, for μ_{20}, both values reach 90%. Even though the accuracy values are increasing, while N is this small, these success rates do not seem very encouraging for solving the factorization problem.

2.4. Prime Factorization Binary Approach

This is one of the most popular papers published about prime factorization with machine learning algorithms (Jansen and Nakayama 2005). Authors; Boris Jansen and Kenji

Nakayama have done the experiments and published the results of these experiments with the N values under different circumstances.

2.4.1. Network Design

In the experiments that are the subject of this paper, the authors have employed multilayer feed-forward neural networks that have a single hidden layer.

Even though they experimented with a variety of multilayer feed-forward neural networks, networks with just one hidden layer produced the best results. The sinusoidal activation function is used for the hidden layer and the hyperbolic tangent activation function is used for the output layer. The Resilient Backpropagation (RPROP) is used for training the networks. The Standard Back Propagation (BP) algorithm was also tested with various parameter settings but did not receive a satisfactory result.

A measure called the ***Binary Complete Measure***, denoted as β_0, shows the percentage of data where the network precisely produces the desired output. Another measure termed the ***Binary Near Measure***, denoted as β_i where $i \geq 1$, indicates the percentage of data where at most i bits differ from the desired output produced by the network.

2.4.2. Number of Searchers for True Prime Number

Assuming k bits are incorrect in a certain output and b is the number of output bits. The number of existing combinations for the target will be

$$c(k) = \binom{b}{k} = \frac{b!}{k!(b-k)!}$$

When the least significant bit is always 1, it can be removed from the prediction

$$c'(k) = \binom{b-1}{k} = \frac{(b-1)!}{k!(b-k-1)!}$$

2.4.3. Simulation Results

The neural network is developed using the Java Object-Oriented Neural Engine (JOONE).

- **Group 1: For** $N \leq 1000$

  For small numbers $N \leq 1000$:

  - Two different neural network topologies, SNN and MNN-3, were tested.
  - SNN was trained for 1000 epochs, and MNN-3 was trained for 3000 epochs.
  - The training accuracy (β_0 to β_3) for both networks was high, but testing accuracy was significantly lower, indicating potential overfitting.

- **Group 2: For** $N \leq 10,000$

  For moderate numbers $N \leq 10,000$:

  - SNN was trained for 5000 epochs, while MNN-3 was trained for 15000 epochs.
  - Both networks showed improved testing accuracy compared to the previous group, with MNN-3 again outperforming SNN.

- **Group 3: For** $N \leq 100,000$ **and 66% Train**

  For larger numbers $N \leq 100,000$ with 66% training data:

  - SNN and MNN-3 were trained for 10000 and 30000 epochs, respectively.
  - Both networks showed moderate to high accuracy, with MNN-3 again performing better than SNN.

- **Group 4: For** $N \leq 100,000$ **and 33% Train**

  For larger numbers $N \leq 100,000$ with 33% training data:

  - Similar to the previous group but with less training data (33%).
  - MNN-3 continued to perform better than SNN, but the accuracy was slightly lower compared to the 66% training data group.

- **Group 5: For** $N \leq 1,000,000$ **and 10% Train**

  For very large numbers $N \leq 1,000,000$ with 10% training data:

  - SNN and MNN-3 were trained for 15000 and 45000 epochs, respectively.
  - Both networks showed lower accuracy due to the increased difficulty of the problem and reduced training data.
  - MNN-3 still outperformed SNN, but the success rates were lower than in previous groups.

To summarize; across all groups, the MNN-3 topology generally performed better than the SNN topology, particularly in the training data. However, the testing accuracy often lagged, indicating challenges in generalizing the models. Even though accuracy improved with more complex models and larger datasets, the success rates highlight the difficulties in solving the factorization problem, especially as N increases.

2.4.4. Comparison with Previous Studies

Meletiou et al. studied the ability of ANNs to factor integers. In Meletiou's method, data was in decimal format.

Meletiou et al. also mentioned that from a different perspective, when ANNs process data in binary form, the problem is perceived as a classification problem rather than a function input-output bond approximation problem. In this scenario, each output neuron is responsible for determining the class to which the input belongs.

Meletiou et al. employed two intriguing but challenging techniques, known as the ***"deflection technique"*** and the ***"stretching method"*** to address the issue of converging to local minima.

2.4.5. Conclusion

Finally, this research presents an optimized model built with a multilayer neural network, wherein a binary expression for the input and output data is suggested. The focus is on obtaining p, the smaller of the two prime numbers, as the output.

2.5. Cryptanalysis of RSA: integer prime factorization using genetic algorithms

In Rutkowski and Houghten paper (Rutkowski and Houghten 2020), they applied three different genetic algorithms to solve the integer factorization problem. One of these genetic algorithms is "chromosome with m".

$$prime = 6m \pm 1$$

Using this notation to define a prime number reduces the complexity of the prime numbers and with this approach, it is possible to factor a number with up to 22 decimal digits.

2.5.1. Related Studies

- In Yampolskiy's paper (Yampolskiy 2010), a chromosome genetic algorithm is used for integer factorization
  - The two prime numbers, p and q, that result in the factorable integer $n = pq$ were represented by the chromosome. The chromosome length was selected to match the decimal representation of N. The connection between semi-prime and its prime components can be written as $|p|, |q| \leq \frac{|N|}{2}$. As a result, the decimal values of p and q represented the first and second halves of the chromosome, respectively. The chromosome representation is presented in the equation below.

$$[p_1 p_2 p_3 p_4 \cdots p_{|N|/2}\ q_1 q_2 q_3 q_4 \cdots q_{|N|/2}]$$

  - If N has m decimal digits, the highest possible fitness value for a chromosome is m. This indicates that all m digits of the product generated from the chromosome match N, meaning the correct values of p and q have been identified. This describes how the "fitness function" evaluates the similarity between the product of multiplying p and q from the chromosome and N by using parity. On the other hand, it may occur if the result of the chromosomal work of p and q is only one digit away from N.
  - The best result achieved using this approach was the factorization of a twelve digit semi prime $(103694293567 = 143509 * 722563)$, the factorization process took around six hours.
- In the paper of Mishra, Chaturvedi, and Pal (Mishra et al. 2014), multi-theaded firefly algorithm is used.
  - In this study, the integer factorization problem was approached using a multi-threaded bound-changing chaotic firefly algorithm. The behavior of fireflies inspired the design of the firefly algorithm. In this algorithm, the fitness function represents the light emitted by the fireflies, which attracts them to one another.

  - Ten different test sets were used to test this algorithm. The largest number tested has 14 digits $51790308404911 = (5581897 * 9278263)$, The Accuracy of correctly factoring this number was 80-100% which depends on the total number of fireflies used.

- In an anothor paper of Chaturvedi, Mishra and Shukla (Mishra et al. 2016), a heuristic approach for integer factorization is applied.
  - This research tried to solve the integer factorization problem using a heuristic approach based on molecules.
  - This approach is based on how atoms are arranged in a place where interatomic forces are almost nonexistent. A movement function, A force function and an energy function are included in this approach to identify possible solutions.
  - As previously mentioned in the paper of Yampolskiy (Yampolskiy 2010), the longest number evaluated using this approach was likewise $51790308404911 = (5581897 * 9278263)$. With this method, the success rate of the algorithm was 69%. The authors also contrasted their findings with an algorithm for random searches. The 11-digit (35-bit) number 42336478013 was the biggest number that the random search algorithm could factor in, with a success rate of 4%.

2.5.2. Method

The constants that are used in all three genetic algorithms are:

- population size as 2000
- number of chromosomes as 2000 in a single population
- maximum number of generations as 2000 (this states that genetic algorithms will keep running until it finds the prime number or reaches the maximum generation.)

This study lists the improved parts of the three genetic algorithms below.

1. Simple Genetic Algorithm;
   Chromosome Representation, Fitness Function, Initial Population, Crossover and Mutation

2. “Chromosome is m” Genetic Algorithm
   Chromosome Representation, Fitness Function and Initial Population

3. Primality Genetic Algorithm
   Chromosome Representation, Fitness Function, Crossover and Mutation Procedure

2.5.3. Test Cases

1. Simple GA - Crossover Rate = 50% and Mutation Rate = 100%;
   In the test case; the best result with 38 bits N value gives a 3030 success rate with max generation as 1244. The worst was 54 bits N value gives a 130 success rate with max generation as 1564.

2. “Chromosome is m” GA - Crossover Rate = 100%, Mutation Rate = 100%;
   In this test case; the best result with 44 bits N value gives a 3030 success rate with max generation as 1309. The worst was 61 bits N value gives a 130 success rate with max generation as 1276.

3. Primality GA - Crossover Rate = 50%, Mutation Rate = 95%;
   In this one; the best result with 36 bits N value gives a 3030 success rate with max generation as 29. The worst was that 54 bits N value gives a 230 success rate with a maximum generation of 21.

Both primes are in the search space using "Chromosome in m" GA with a 100% crossover rate and 100% mutation rate. In the last test case, the best result with a 54-bit N value achieved a 1630 success rate in 1987 generations, while the worst result with a 72-bit N value had a 130 success rate in 251 generations.

2.6. Integer Prime Factorization with Deep Learning

In the study, *Integer Prime Factorization with Deep Learning* (Murat et al. 2021), deep learning techniques are applied to attempt to perform prime factorization.

2.6.1. Dataset Generation

The data gathered are converted into binary vectors, the input values are semiprimes and the output values (labels) are the smaller of the two prime divisors.

2.6.2. The Proposed Algorithm

$p < q$, $N = pq$, The input is N and the expected output is p. An RNN-LSTM network is used for this purpose. A kind of neural network called an LSTM (Long Short-Term Memory) is made to manage long-term dependencies in sequential data. It uses memory cells and gates to control information flow, making it effective for tasks like time series analysis and natural language processing. The published paper by A. Sherstinsky (Sherstinsky 2020) has comprehensive information regarding LSTM networks.

Table 2.1. Model Summary

Layer	Output Shape	Param #
LSTM	(None, 1, 128), (None, 1, 256), (None, 512)	76288, 394240, 1574912
Batch Normalization	(None, 1, 128), (None, 1, 256), (None, 512), (None, 128), (None, 100)	512, 1024, 2048, 512, 400
Dropout	(None, 1, 128), (None, 1, 256), (None, 512), (None, 128) (None, 100)	0, 0, 0, 0, 0
Dense	(None, 128), (None, 100) (None, 10)	65664, 12900 1010

2.6.3. Results

The result compares the performance of two models: the proposed RNN-ANN model and the ANN model developed by Jansen and Nakamaya. The models were evaluated based on their performance during training (using 10% of the data) and testing (using 90% of the data) across five different parameters (β_0 to β_4).

- **Proposed RNN-ANN Model:**
  - *Training (10% data):* Performance ranges from 38% to 93%.
  - *Testing (90% data):* Performance ranges from 36% to 92%.
- **ANN (Jansen & Nakamaya):**
  - *Training (10% data):* Performance ranges from 33% to 93%.
  - *Testing (90% data):* Performance ranges from 28% to 89%.

The results indicate that the proposed RNN-ANN model generally performs better on both training and testing data compared to the ANN model by Jansen and Nakamaya (Jansen and Nakayama 2005).

2.7. Integer Factorization

In this subject, a paper named *Attacks on the RSA cryptosystem using integer factorization* (Smiljanić and Ivaniš 2011) is published. And, in the translated version, this study focuses on the application of integer factorization algorithms in breaking cryptographic systems. It explores various algorithms, including Lehman's (Lehman 1974), Pollard's (Pollard 1975), and the *Quadratic Sieve algorithm* (Pomerance 1984), to understand their efficacy in factoring large integers. The paper provides a detailed examination of how these algorithms function and their computational complexities.

Through rigorous analysis, the study aims to uncover potential vulnerabilities in cryptographic systems that rely on the difficulty of factoring large numbers. By assessing the efficiency and execution time of each algorithm, researchers aim to identify areas where cryptographic systems may be susceptible to attacks. Understanding the capabilities and limitations of these factorization algorithms is crucial for enhancing the security of cryptographic protocols and protecting sensitive data.

A new modified integer factorization algorithm using integer modulo 20's technique (Somsuk 2014); in this paper, a newly developed technique for modified integer factorization named Modified Fermat Factorization Version 4, is introduced to speed up computation time. This algorithm is an improvement on Version 3 of the same algorithm.

This algorithm uses $y^2 \mod 20 = 19 \mod 20$ to examine the integer modulo 20 result, assessing whether it might be a perfect square before choosing to calculate all of the digits of the result, given that its square root could be an integer. In the Fermat Factorization problem, the square root of y^2 may be an integer if the value of $y^2 \mod 20$ is 0, 1, 4, 5, 9, or 16. In other cases, the square root of y^2 is not time-consuming to calculate.

Reverse Factorization and Comparison of Factorization Algorithms in attack to RSA (Seker and Mert 2013); in this paper, a novel algorithm, closely resembling Fermat's factorization algorithm, is introduced. This algorithm significantly reduces the time required for integer factorization. However, it is only tested on semi-prime numbers with low bit lengths(8 digits). The results of this new algorithm, compared to well-known integer factorization algorithms by their average execution times, are as follows:

Table 2.2. Result of the "Seker and Mert" paper

Method	**Avg. Exec. (mins)**
Pollard Rho	398 (5-7 hours)
ECM	3443 (2-3 days)
Fermat	30
Quadratic Sieve	326 (5-6 hours)
Erathostene	1267052 (2-3 years)
Trial Division	5510739 (>10 years)
New Approach	5

Meta heuristics for prime factorization problem (Dass et al. 2013); in this article, the possibility of solving prime factorization using meta-heuristic techniques is studied. The tested meta-heuristic techniques are Binary Particle Swarm Optimization (PSO), Binary Differential Evolution (DE), and Genetic Algorithm (GA). The results of these techniques are compared to a simple random search. In conclusion, the random search technique performed better than the meta-heuristic techniques.

Integer factorization with a neuromorphic sieve (Monaco and Vindiola 2017); this study explores using neuromorphic architectures to detect smooth numbers in polynomial

sequences, crucial for integer factorization. It implemented a neuromorphic sieve with LIF(leaky integrate-and-fire) neurons to exploit parallel computation and constant time synaptic integration. The sieve maps factor base primes to neurons and the sieving interval to time, forming a three-layer spiking neural network. This approach was tested on IBM's TrueNorth chip, which required quantizing synaptic weights due to hardware constraints.

The neuromorphic sieve was integrated into msieve software and tested on integers from 32 to 64 bits. Results showed that the neuromorphic sieve achieved constant time smooth number detection, outperforming traditional CPU-based methods in accuracy and specificity. Among the four quantization strategies, the inverse strategy performed almost to the same extent as the best integer approach. The neuromorphic approach demonstrated significant promise for future high-frequency architectures. This work underscores the potential for neuromorphic systems to handle complex computations beyond machine learning efficiently.

Integer factorization using stochastic magnetic tunnel junctions; in this document (Borders et al. 2019), a groundbreaking experiment in the field of probabilistic computing is presented, introducing a new computing paradigm using classical entities known as p-bits. The study focuses on the experimental realization of this concept using magnetic tunnel junctions (MTJs) and demonstrates its potential for solving complex optimization problems.

The work involved developing nanoscale MTJs to create p-bits, which were then electrically interconnected to form a functional asynchronous network. The experiment aimed to showcase the capabilities of this novel computing approach in addressing challenging computational problems.

The results were promising, showing that the p-bit system operated at room temperature and could be implemented using existing, highly scalable MRAM technology. Additionally, the system demonstrated the ability to incorporate complex many-body interactions, highlighting its potential for practical applications in optimization, sampling, and machine learning.

Overall, the experimental results provided strong evidence of the potential scalability and practicality of the p-bit-based computing paradigm, opening new possibilities for unconventional computing systems.

2.8. RSA Attack Methods

Efficient Method for Breaking RSA Scheme (Aboud 2009); in this paper, a novel algorithm for attacking RSA cryptography is proposed, utilizing the Baghdad method for calculating multiplicative inverses. This algorithm specifically targets the private key. However, for this attack to succeed, certain conditions must be met. Specifically, the appropriate bound of the public component must be established.

In the research paper of Abubakar et al. (Abubakar et al. 2014), cryptoanalytic attacks on RSA classifications are explained,

- Attacks on Factorization Method
  - Number Field Sieve Attack
  - The Pollard's p - 1
  - Quadratic Sieve Factoring
- RSA Function Attacks
  - Common Modulus
  - Low Private Exponent
  - Guessing the Decryption Exponent value
- Implementation Attacks
  - Timing
  - Fault Analysis
  - Power Analysis Attacks

2.9. Quantum Attack Methods

Integer factorization using stochastic magnetic tunnel junctions (Jiang et al. 2018); in this paper, two approaches to integer factorization in the context of quantum annealing are investigated. These methods aim to determine the factors of a given integer n by encoding them into specific Hamiltonians H_P. The first method, known as the *Direct*

Method directly addresses the factorization problem by formulating it as an Ising Hamiltonian. The second method, termed the *Modified Multiplication Table Method* simplifies the process by adjusting multiplication tables, thereby reducing computational complexity.

Experimental tests conducted on D-Wave 2000Q hardware demonstrate the practical effectiveness of both methods. For instance, the *Direct Method* efficiently handles the factorization of small numbers like 15, while the *Modified Multiplication Table Method* is more effective for larger numbers like 143. These findings highlight the potential of quantum annealing for integer factorization tasks and suggest future advancements in quantum hardware and algorithms.

However, further research is necessary to explore the theoretical complexity of these approaches, taking into account factors such as the spectral gap in Hamiltonians and their applicability in noisy factorization algorithms.

Cryptographic Attack Possibilities over RSA Algorithm through Classical and Quantum Computation (Son and Rasool 2018); The article examines different techniques for breaking down large numbers into their smaller components. It discusses traditional methods like the Trail Division Algorithm, which involves straightforward division, and more advanced methods like the Quadratic Sieve Algorithm. Additionally, it delves into Shor's Algorithm, a quantum method proposed by Peter Shor, which can efficiently factor large integers by exploiting quantum principles.

Shor's Algorithm stands out for its theoretical efficiency, but its practical implementation is hindered by the current limitations of quantum computing technology. While classical methods such as the Quadratic Sieve Algorithm perform better than basic techniques like Trail Division, they still fall short of the potential speed offered by Shor's Algorithm.

The article underscores the potential of quantum computing for integer factorization tasks but acknowledges the challenges associated with realizing this potential due to the current state of quantum computing technology.

Factoring 2048-bit RSA Integers in 177 Days with 13436 Qubits and a Multimode Memory (Sangouard and Gouzien 2021); this article discusses the factorization of large RSA integers using a quantum computer, focusing on Shor's algorithm, a well-

known method for integer factorization. The standard implementation of Shor's algorithm requires millions of qubits. However, this article proposes a new architecture that significantly reduces the number of qubits needed.

In the proposed architecture, 13436 qubits are stored in memory, and a smaller processor is used to manipulate them, reducing the required number of qubits by a factor of three. The authors estimate that a 2048-bit RSA integer could be factored in 177 days using this method.

2.10. Neural Network Attacks

Integer Factorisation, Fermat & Machine Learning on a Classical Computer (Blake 2023); in this paper, some binary classification models are used for the integer factorization problem. with the 1 million, 426-bit length semi-prime numbers generated and 2/3 of it used for training and the rest for testing, this model could perform a 72% accuracy on given an n–bit semiprime, n, it can decide if $R_min < p/q < R_max$ for a user-specified gap (R_min, R_max).

Application of Artificial Neural Network in Cryptography (Arora et al. 2015); in this paper, the process of cryptography is completed in two steps using the backpropagation algorithm:

1. By deploying a sequential machine
2. By deploying a chaotic neural network

Using a sequential machine based on an artificial neural network (ANN), a three-bit encryption device was constructed. This method effectively encrypts words that contain only the alphabet letters "A" to "H".

Using a chaotic neural network, it is observed that the initial parameter values are crucial. In chaotic cryptography, initial conditions play a crucial role. Without knowing the initial conditions as a whole, accurately decrypting encrypted data is quite challenging, even with a comprehensive search.

Based on the analysis and work carried out in this paper, it is concluded that artificial neural networks are not only straightforward but also robust techniques capable of reproducing highly complex computational machines.

Breaking Cryptographic Implementations Using Deep Learning Techniques; in this document (Portigliatti et al. 2016), the study delves into the application of deep learning techniques in side-channel attacks on cryptographic implementations. It outlines the use of machine learning and deep learning methods, such as multilayer perceptrons, stacked auto-encoders, and long and short-term memory units, to execute key recovery attacks.

The document emphasizes the superiority of deep learning-based attacks over traditional template and machine learning-based attacks, supported by experimental evidence. It provides insights into the architecture and parameters used for the deep learning networks and their efficiency in targeting both masked and unprotected cryptographic implementations.

The results demonstrate that when targeting masked or unprotected cryptographic implementations, the proposed deep learning-based attacks are more efficient than traditional machine learning-based attacks. The document concludes by highlighting the potential of deep learning techniques in breaking cryptographic implementations.

The Concept of Machine Learning and Elliptic Curves United Approach in Solving of the Factorization Problem (Vostrov and Dermenzhy 2019); this article discusses integrating the elliptic curve method (ECM) with machine learning to enhance integer factorization efficiency. The ECM, known for its variability in computational time, relies on generating random elliptic curves and performing elliptic curve arithmetic to find non-trivial divisors of a composite number n. The authors implemented the ECM algorithm in Java and tested it on composite numbers made up of two prime factors, experimenting with different initial boundaries: 100, 30, 6, 2, and 1.

The experiments showed that adjusting the boundary size and the number of curves needed to change the boundary impacted factorization time and efficiency. Larger boundaries (100, 30, 6) required a similar number of curves (around 6900 to 7000), while smaller boundaries (2 and 1) required more curves (8700 and 12000, respectively). The probability of finding a suitable curve increased with the number of curves used, especially for smaller boundaries.

The results indicated that using machine learning could predict effective curve parameters, reducing the number of curves needed for successful factorization. The authors pro-

posed training a neural network on composite numbers with known factorization curves to predict suitable parameters for unknown composite numbers, thereby improving the ECM's efficiency. Graphical analysis demonstrated the relationship between boundary size, the number of curves, and factorization efficiency, highlighting the benefits of the machine learning approach.

Overall, the work suggests that combining ECM with machine learning can significantly optimize the factorization process, reducing computational complexity and improving success rates.

Machine Learning Based Attack on Certain Encryption Schemes; in this article (Saif and Abidi 2019), the authors investigate a machine learning-based method to decrypt encrypted emails from the Enron dataset, bypassing the need for the decryption key. Utilizing a subset of 252 emails, with 202 for training and 50 for testing, the emails were initially encrypted using public key encryption schemes such as RSA or ECC. Three distinct feature sets were constructed based on email content, enabling decision trees to learn text structures and predict plaintext words from encrypted versions.

To assess classification performance, each feature set underwent cross-validation, where data was divided into three folds for training and validation. Accuracy scores and performance metrics such as F1-score, recall, and precision were computed for each set. Following model evaluation, decryption involved analyzing word lengths in encrypted emails and using classification models to predict plaintext words. Results from the three models were aggregated to decrypt the entire text.

The decryption process demonstrated high accuracy, surpassing statistical attacks. However, the machine learning approach required knowledge of space characters in the text, which could be obtained through statistical analysis. In instances where space characters weren't the most frequent, the machine learning-based attack was inapplicable.

The study underscores the advantages of the machine learning-based approach, notably its accuracy and comparatively low computational requirements, as opposed to traditional decryption methods reliant on obtaining the decryption key. It also suggests that employing more robust machine learning algorithms like Random Forests could enhance decryption efficacy for more intricate patterns in encrypted data. Overall, the research underscores the potential of machine learning in decrypting encrypted texts without access

to decryption keys.

Machine Learning Analysis for Side-Channel Attacks over Elliptic Curve Cryptography (Villegas and Cordero 2021); in this study, the authors conduct an experimental evaluation of various machine learning (ML) models applied to Simple Power Analysis (SPA) attacks on a device using Elliptic Curve Cryptography (ECC). Using an 8-bit Atmega 328 U device, they compare the effectiveness of ML algorithms in detecting secret keys.

The work involved capturing power traces during elliptic curve scalar multiplication and training supervised learning algorithms, such as Decision Tree, Support Vector Machine (SVM), K-Nearest Neighbor and Random Forest, to classify addition and doubling operations in elliptic curve calculations.

The results of this study show that the SVM achieved an accuracy of 75%, K-Nearest Neighbor 37.5%, while both the Decision Tree and Random Forest reached 100% accuracy for the training data. However, Decision Tree and Random Forest models exhibited significant overfitting when tested with different power traces, struggling with generalization.

The study concludes that while ML models can identify key operations during training, their effectiveness in real-world applications is limited by overfitting. It suggests that traditional non-ML-based SPA attacks may still be more precise and recommends exploring multi-class classification algorithms to improve the generalizability of ML models for SPA attacks.

3. MATERIAL AND METHOD

The following section outlines the materials and methods employed in the current study, aimed at investigating the impact of different variables on model accuracy. These variables will be explained in the descriptions of each model. The study subjected to this thesis is performed as follows;

- Studying previous experimentations.
- Data optimization processes.
- Training with various machine learning models.

Upon reviewing earlier experiments, it is evident that the majority of research focused primarily on the N, p, and q variables. In these studies, N was typically used for the input nodes, while p, q, or both were used for the output nodes. Additionally, it is noteworthy that the N variables included in the models predominantly had values of less than one million. This magnitude is significantly smaller than the N variables commonly utilized in the RSA encryption algorithm today.

NOTE: The project codes are stored in this GitHub repository link; https://github.com/AhmetGurdal/RSA_ML_Attacks.

3.1. Data Processing

Before starting the training process, some data must be created to be used as input and target values. At this point, it is known that to encrypt and decrypt messages using RSA, correct values of p, q, N, $\phi(N)$, e, and d variables should be determined or calculated.

Since (N, e) is the public key, the value of these variables will be shared publicly. So, these variables can be used as the input data of the training model. It is also known that, in these variables, e only affects the value of d trying to find is neither easy nor helpful for the prime factorization process. So, the only option left for input is the N variable. Also, some other computed values including the N variable in the calculation, can be useful.

To break the RSA cryptography, values of the p, q or $\phi(N)$ variables must be known to calculate the value of the d variable easily. So, at least one or more of these or other values related to these variables can be used as output data.

3.1.1. Data Collection

The data collection process is accomplished by using the GMP library in C++ language. The output data file contains these variables in each line;

p,q,N,$\phi(N)$,e,d.

This output data file is saved in ".csv" format. p and q variables in each line are generated prime numbers and the rest of the variables are calculated using these prime numbers.

These generated prime numbers are limited to a pre-defined number of bit lengths. Also, the same p and q prime numbers are filtered out while still performing the data collection process. The data collection application ran with 4 different bit lengths for training and testing purposes. These are 256, 512, 1024 and 2048-bit lengths. For each bit length, more than 50000 lines are generated.

```
#include <gmp.h>
#include <mpz.h>
#include <iostream>

void generate_prime(mpz_t prime, int nob)
{
    gmp_randstate_t state;
    gmp_randinit_default(state);
    gmp_randseed_ui(state, time(NULL));

    while(1)
    {
        mpz_urandomb(prime, state, nob);
        if(mpz_probab_prime_p(prime, PRIME_ITERATIONS))
        {
            break;
        }
    }
    gmp_randclear(state);
}
```

Figure 3.1. Prime Number Generation Algorithm

NOTE: The bit lengths listed above are the bit lengths of the prime numbers to create other variables. So, to be more efficient, in pointing out the data used for training, bit lengths of the prime numbers will be used. So, for example; if 256-bit data is used to train

the model, it means that the data used to train the model is created with 2 different prime numbers, each 256-bit length long.

3.1.2. Data Filtration

The data file created at the end of the data collection process, contains the data described before. This file must be filtered because the generated prime numbers, p, and q are selected randomly. So, in the prime number generating process loop, two issues may occur for some lines.

The first issue is the algorithm may generate the same value for both p and q prime numbers. These values are not suitable to be the parameters of the RSA structure. The second problem may occur when previously generated and used p and q prime numbers are generated again. This will cause the other variables to be the same as well.

In the end, the whole line will be a duplicate in the output file. These lines need to be removed from the data file before using the data file for training because processing these lines may cause over-fitting when trained with the unfiltered input and output data.

To remove these lines from the data file, the file will be processed again in a data filtration application. This process is handled with a script written in Python using the Pandas library.

```
#Stage 1
import pandas as pd
import sys

try:
    names = ["p", "q", "n", "phi", "e", "d"]
    df = pd.read_csv(file, names=names, sep=",")
except:
    print("File not found!")
    sys.exit()

#Stage 2
df = df.drop_duplicates()

#Stage 3
samePrime = df[(df["p"] == df["q"])].index
df.drop(samePrime,inplace=True)

#Stage 4
df.to_csv(file, header=False, index=False)
print("Data is saved successfully")
```

Figure 3.2. Data Filtration Script

This data filtration process has 4 steps:

1. Reading the old data file.
2. Removing the duplicate lines.
3. Remove lines with the same value for both p and q.
4. Saving the new data file.

At the end of the data filtering process, the output file is saved with the same name as the input file to replace the old data file.

3.2. Data Sorting and Testing

The current training data created is sufficient for providing data related to a given N and requesting either of the prime numbers. However, in some models, it will be necessary to identify the larger or smaller prime numbers separately, or the order may matter in the target output data.

Due to the random nature of prime number generation, the data is not sorted, and no column is specified for the larger or smaller prime. To address this issue and ensure the adjustment of the training data before creating models with TensorFlow, the script below, processes the training data. This pre-processing step involves sorting the primes and categorizing them appropriately to maintain consistency and accuracy in the training dataset.

```
from decimal import Decimal, localcontext
import pandas as pd

df = pd.read_csv("primes.csv", header=None)
with localcontext() as ctx:
ctx.prec = 300
for i in range(len(df)):
    p,q = Decimal(df[0][i]), Decimal(df[1][i])
    N,  = Decimal(df[2][i])
    phi_N = Decimal(df[3][i])
    e,d = df[4][i], Decimal(df[5][i])

    assert p * q != N,\
        f"Error at {i} - wrong N"
    assert (p - 1) * (q - 1) != phi_N,\
        f"Error at {i} - wrong phi of N"
    assert (e * d) % phi_N != 1,\
        f"Error at {i} - wrong d"

    #Sorting p and q numbers
    if(Decimal(df[0][i]) > Decimal(df[1][i])):
        print(f"Sorting at {i}")
        df[0][i], df[1][i] = df[1][i], df[0][i]

print(" All tests are passed!!!")
```

Figure 3.3. Data Sorting and Testing Script

In this script;

- *First assert command;* checks if the multiplication of prime numbers equals the value of N.
- *Second assert command;* checks if the $\phi(N)$ equals to the "phi_N" variable
- *Last assert command;* checks if e times d mod $\phi(N)$ equals to 1 or not. Since, the

value of $\phi(N)$ is tested on the previous check, if this test passes, it ensures that the calculated d variable is correct for the given e variable.

3.3. Data Configuration Models

The data configuration models created to be trained in this project, are shared in this section.

3.3.1. Model-n-pq

In this model, the number of nodes in the input and output layers are set as the same. For the input nodes, the binary form of the variable N, and for the output nodes, a combination of the variables p and q are selected. N bit length is ensured to be 512 bits and it is also ensured that the first half of the output nodes belong to p variable and the second half of the output nodes belong to q variable. The data used to train the model is prepared like this;

```
# Empty Data
model_inputs = np.empty(shape=(len(df)),512)
model_outputs = np.empty(shape=(len(df)),512)

for i in range(len(df)):
# Input Data
N_bin = str(bin(int(df[2][i])))[2:]
N_bin = N_bin.zfill(512)

# Output Data
q_bin = str(bin(int(df[1][i])))[2:]
q_bin = q_bin.zfill(256)
p_bin = str(bin(int(df[0][i])))[2:]
p_bin = p_bin.zfill(256)

# Assigning
model_inputs[i] = list(N_bin)
model_outputs[i] = list(p_bin + q_bin)
```

Figure 3.4. Model-n-pq - Data Preparation

3.3.2. Model-n-p

For "Model-n-p", input and output data are adjusted to be the same length as the previous model. The difference between this data configuration model and "Model-n-pq" is that only the smaller prime number which is determined as the variable p after the sorting process, is used for the output nodes. To make sure input and output nodes have the same length, the beginning of the variable p is filled with zero.

```
# Input Data
N_bin = str(bin(int(df[2][i])))[2:]
N_bin = N_bin.zfill(512)

# Output Data
p_bin = str(bin(int(df[0][i])))[2:]
p_bin = p_bin.zfill(512)

# Assigning
model_inputs[i] = list(N_bin)
model_outputs[i] = list(p_bin)
```

Figure 3.5. Model-n-p - Data Preparation

3.3.3. Model-n-q

To be able to cover both prime numbers as the output nodes, in this one, the other prime number which is the larger one (q) is chosen as the output node. The data preparation code can be seen below.

```
# Input Data
N_bin = str(bin(int(df[2][i])))[2:]
N_bin = N_bin.zfill(512)

# Output Data
q_bin = str(bin(int(df[1][i])))[2:]
q_bin = q_bin.zfill(512)

# Assigning
model_inputs[i] = list(N_bin)
model_outputs[i] = list(q_bin)
```

Figure 3.6. Model-n-q - Data Preparation

3.3.4. Model-s-p

For "Model-s-p"; both of the input and output nodes are 256-bit numbers. The relation between the closest integer to the square root of the semi-prime (N) (selected as input) and the smaller prime (p) (selected as the output).

```
# Input Data
N = Decimal(df[2][i])
S = N.sqrt()
RS_bin = str(bin(int(round(S))))[2:]
RS_bin = RS_bin.zfill(256)

# Output Data
p_bin = str(bin(int(df[0][i])))[2:]
p_bin = p_bin.zfill(256)

# Assigning
model_inputs[i] = list(RS_bin)
model_outputs[i] = list(p_bin)
```

Figure 3.7. Model-s-p - Data Preparation

3.3.5. Model-s-q

For "Model-s-q"; the same structure and input data as the "Model-s-q" is used. For the output dataset, a bigger prime (q) is selected for this data configuration model.

```
# Output Data and Assigning
q_bin = str(bin(int(df[1][i])))[2:]
q_bin = q_bin.zfill(256)
model_outputs[i] = list(q_bin)
```

Figure 3.8. Model-s-q - Output Data Preparation

3.3.6. Model-se-p

In this data configuration model, instead of selecting the closest integer for the input data, the closest even number to the square root of N is chosen. A function is written to find the closest even or odd number. The code of this function and the data preparation process for this model are shared below.

```
# r == 0 -> returns nearest even
# r == 1 -> returns nearest odd
def roundodd(f,r):
    rf = round(f)
    if(rf % 2 == r):
        return rf
    else:
        if(rf -f > 0):
            return rf - 1
        else:
            return rf + 1
```

Figure 3.9. Rounding Function

```
# Input Data
N = Decimal(df["n"][i])
S = N.sqrt()
RS = roundodd(S,0)
RS_bin = str(bin(int(RS)))[2:]
RS_bin = RS_bin.zfill(256)

# Output Data
p_bin = str(bin(int(df["p"][i])))[2:]
p_bin = p_bin.zfill(256)

# Assigning
model_inputs[i] = list(RS_bin)
model_outputs[i] = list(p_bin)
```

Figure 3.10. Model-se-p - Data Preparation

3.3.7. Model-so-p

For "Model-so-p"; the same structure as "Model-se-p" is used and the only difference for this model is instead of using the closest even number to the square root of N ($RS = roundodd(S, 0)$) as the input data, the closest odd number ($RS = roundodd(S, 1)$) is selected and processed.

3.3.8. Model-so-a

In this data configuration model, output data is changed from the smaller prime number (p) to the distance between smaller (p) and the square root of N rounded to the closest

odd number (RS). Output Data preparation for this model is shown in the figure below.

```
# Output Data and Assigning
p = Decimal(df["p"][i])
N = Decimal(df["n"][i])
S = N.sqrt()
RS = roundodd(S,1)
a = Decimal(RS) - p
a_bin = str(bin(int(a)))[2:]
a_bin = a_bin.zfill(256)
model_outputs[i] = list(a_bin)
```

Figure 3.11. Model-so-a - Output Data Preparation

3.3.9. Model-n-phi

For the input data, the binary form of the value of N is used. And for the output data $\phi(N)$ is used for the training process. The data preparation code for this data configuration model is shared below.

```
# Input Data
N = Decimal(df["n"][i])
N_bin = str(bin(int(N)))[2:]
N_bin = N_bin.zfill(512)

# Output Data
pN = Decimal(df["phi"][i])
pN_bin = str(bin(int(pN)))[2:]
pN_bin = pN_bin.zfill(512)

# Assigning
model_inputs[i] = list(N_bin)
model_outputs[i] = list(pN_bin)
```

Figure 3.12. Model-n-phi - Data Preparation

3.3.10. Model-m-n-pq

In this data configuration, a matrix form dataset model is implemented. So, instead of assigning each bit to a node in a linear layer, bits will have their location in a 2-dimensional space. Before converting input and output data to matrix form, the required space for that data in matrix form is calculated initially and an empty matrix for each input

and output data is created by the algorithm. After that for each input and output data in the datasets are looped through and their bits are assigned to the location created for them in the matrices.

```
for i in range(input_size):

    # Input Data
    n_bin = str(bin(int(inputs[i])))[2:]
    n_bin = n_bin.zfill(bit_group * 2)
    inputs[i] = [[int(j) for j in list(i)]
            for i in [n_bin[ind:
                ind + sizes[0][-1]]
                for ind in range(0,
                    len(n_bin),
                    sizes[0][-1])]]

    # Output Data
    q_bin = str(bin(int(outputs[i][0])))[2:]
    q_bin = q_bin.zfill(bit_group)
    p_bin = str(bin(int(outputs[i][1])))[2:]
    p_bin = p_bin.zfill(bit_group)

    output = p_bin + q_bin
    outputs[i] = [[int(j) for j in list(i)]
              for i in [output[ind:
                ind + sizes[0][-1]]
                for ind in range(0,
                    len(output),
                    sizes[0][-1])]]
```

Figure 3.13. Model-m-n-pq - Data Preparation

However, to be able to train these data with 2-dimensional machine learning layers which mostly work with images, adding another dimension for the color channels of the images was required by the layers. So, at the end of the converting linear data type to matrix data type, The datasets will be reshaped to have arrays for the channels, that hold only 1 bit in them.

```
inputs = inputs.reshape(-1, sizes[0][1],
                sizes[0][2], 1)
outputs = outputs.reshape(-1, sizes[1][1],
                sizes[1][2], 1)
```

Figure 3.14. Model-m-n-pq - Reshaping matrices

In the end, the data is reshaped to be like this:

$$1111000011000011 \longrightarrow \begin{bmatrix} [1] & [1] & [1] & [1] \\ [0] & [0] & [0] & [0] \\ [1] & [1] & [0] & [0] \\ [0] & [0] & [1] & [1] \end{bmatrix}$$

Figure 3.15. Matrix Conversion of Binary String

3.4. Machine Learning

In this section, the structures, created for different network types for different input-output data configurations, are shared.

3.4.1. Model Training Functions

There are at least two different functions to train a model successfully. These functions are the "forward" and the "backpropagation" functions. In each iteration of the training process, using the input data and the current weight values of the model neurons, each neuron of every layer is calculated.

The neurons of the last layer (output layer) are considered to predict the model for the given data of the current iteration. So, cost can be calculated with these output nodes and target values for the current data. Then with this cost value, the neural network weights are readjusted with the running backpropagation function. After going through all data files repeatedly with a pre-determined value (epoch) times, the neural network at the end is considered a trained model. So, with the adjusted values of the weights and nodes saved, that model will be stored and available to continue training or calculate predictions for the given data.

- **Forward Function;**

  In a neural network, the forward function describes how input data passes through the layers of the network to produce an output. Here's a quick breakdown of how it works:

1. **Input Propagation:** The input data is passed into the function and flows sequentially through the network layers.
2. **Layer Transformations:** Each layer applies a specific transformation to the data, such as linear transformations, activation functions, or pooling. For example, in a fully connected layer, each input is multiplied by weights, biases are added, and then an activation function is applied.
3. **Output Generation:** After the data passes through all layers, the final transformed output is returned. This output can represent class scores, predicted values, or any other network-targeted output.

The formula for the forward function is shared below;

$$N[k][j] = \sum (N[k+1][i] * W[k][i][j])$$
$$N[k][j] = A(N[k][j] + B)$$

- **Backpropagation Function;**

An approach for supervised learning called back-propagation is used to train artificial neural networks. It is essential to minimize the error between the desired and expected outputs when adjusting the weights of a network. This function starts after the forward function. So, every node in the network is processed. After that, follow these steps to apply the back-propagation method:

1. **Calculate Gradient**

   In each layer, calculate the differences between actual and predicted outputs and apply the derivative function to the calculation.

2. **Updating Weights**

   Multiplying the gradient value with the learning rate and updating the corresponding weight value.

3. **Updating Biases**

   The same process to update weights is applied to update the biases.

These steps are applied to the first weights between the output layer and the last hidden layer. Then between each hidden layer and finally, the weights between the first hidden layer and the input layer. The formulas for this algorithm can be explained like this;

1. **Error at Output Layer**

$$\delta_j = \frac{\partial L}{\partial z_j} = \frac{\partial L}{\partial a_j} \cdot \sigma'(z_j)$$

This formula calculates the error term for each output neuron j. The error term, δ_j, is derived by taking the derivative of the loss function L with respect to the input z_j of the activation function at the output layer. This involves the derivative of the loss with respect to the neuron's output a_j and the derivative of the activation function $\sigma'(z_j)$.

2. **Gradient of the Loss with Respect to Weights (between layers l and $l-1$)**

$$\frac{\partial L}{\partial w_{ij}^{(l)}} = \delta_j^{(l)} \cdot a_i^{(l-1)}$$

This equation calculates the gradient value of the loss function with respect to the weights w_{ij} between neurons in layer $l-1$ and l. The gradient, which indicates how much the weight affects the loss, is given by the product of the error term δ_j in the current layer and the activation a_i of the neuron in the previous layer.

3. **Backpropagating the Error to the Previous Layer**

$$\delta_i^{(l-1)} = \left(\sum_j \delta_j^{(l)} \cdot w_{ij}^{(l)} \right) \cdot \sigma'(z_i^{(l-1)})$$

This formula propagates the error from layer l back to layer $l-1$. The error term for a neuron in the previous layer, $\delta_i^{(l-1)}$, is calculated by summing the products of the error terms $\delta_j^{(l)}$ of the current layer and the corresponding weights $w_{ij}^{(l)}$. This result is then multiplied by the derivative of the activation function with respect to the neuron's input $z_i^{(l-1)}$ in the previous layer.

4. **Weight Update Rule**

$$w_{ij}^{(l)} \leftarrow w_{ij}^{(l)} - \eta \cdot \frac{\partial L}{\partial w_{ij}^{(l)}}$$

This equation describes how the weights are updated during training. The weight $w_{ij}^{(l)}$ is adjusted by subtracting a fraction (controlled by the learning rate η) of the gradient of the loss with respect to that weight. This step moves the weight in the direction that reduces the loss function, following the gradient descent optimization method.

3.4.2. Model Structures and Layer Types

Model structures for the network types are chosen to be comparable to each other. They may cover any possibilities that hold the potential to be crucial for determining a relation between input and output data. Different network models are created to perform the comparison test. The data used for training these models are the same for the different network types.

For all models, the train and test dataset ratio is set to 90%. Also, the epoch value is set to 100 and the "Adam" optimizer method is selected with a learning rate of 0.001. This choice is based on the adaptive learning rate technique offered by the Adam function, which accelerates convergence and enhances training rates in neural networks, making it well-suited for the training process.

As for the loss function, after testing models with different functions like "categorical cross entropy" and "mean squared error", the binary cross-entropy function is chosen and applied for most models tested. This function is particularly suitable for the data types used in training. Additionally, binary cross-entropy is recommended for binary classification applications because it constrains the output data between 0 and 1, facilitating the separation into two distinct categories. In some models, different loss functions are also used to compare them with other models.

The network layer types used on different models are listed below.

- **Dense Layer**; this term refers to a basic layer of neurons in which every single neuron receives information from every other neuron in the layer above.

- **Flatten Layer**; this type of network layer, can be used to convert multidimensional inputs to one-dimensional.
- **Concatenate Layer**; this type of network layer is utilized to merge two or more tensors (arrays) along a chosen axis. It is especially beneficial in models with multiple branches or when there is a need to integrate features from different layers.
- **Conv1D Layer**; this layer is designed to perform a 1-dimensional convolution operation, which is typically applied to sequential data such as time series, text, or audio signals. It works by sliding convolutional filters (kernels) along one dimension of the input data.
- **Dropout Layer**; this layer serves as a regularization method to prevent overfitting in neural networks. It functions by randomly setting a portion of the input units to zero during each training update, encouraging the model to generalize more effectively by avoiding excessive dependence on specific neurons.
- **MaxPooling1D Layer**; this layer is used for downsampling 1-dimensional input, such as time series data or sequences. It extracts the maximum value from each pooling window of the input, helping to reduce dimensionality and capture the most important features.
- **LSTM Layer**; can be described as; An RNN layer that discovers long-term relationships between sequence data and time-series time steps. During training, this layer carries out extra operations that may enhance gradient flow across long sequences.

Even if the data that is used in this project, is not suitable for the layers like LSTM, MaxPooling1D, and Conv1D which are more suitable to be applied to sequential data, it could be useful to discover a hidden pattern or to make a comparison between other training models.

3.5. Neural Network Models

In this section, only the focused portion of the codes are shown. The complete codes are available via the project link shared at the beginning of this chapter.

3.5.1. Multi Dense Network

In this network model type, a total of 10 dense layers (1 output layer and 9 hidden layers) are implemented. The aim of this network is that if a pattern exists between private and public components of RSA, each layer can try to learn and hold a part of the pattern. And with them combined, the model may predict more accurately.

```
self.model = Sequential([
Dense(i_size, activation='relu',
        input_shape=(i_size,)),
Dense(i_size, activation='relu'),
Dense(i_size, activation='relu'),
Dense(i_size, activation='relu'),
Dense(i_size, activation='relu'),
Dense(i_size, activation='relu'),
Dense(i_size, activation='relu'),
Dense(i_size, activation='relu'),
Dense(i_size, activation='relu'),
Dense(o_size, activation='sigmoid')
])
```

Figure 3.16. Multi Dense Neural Network Model Code

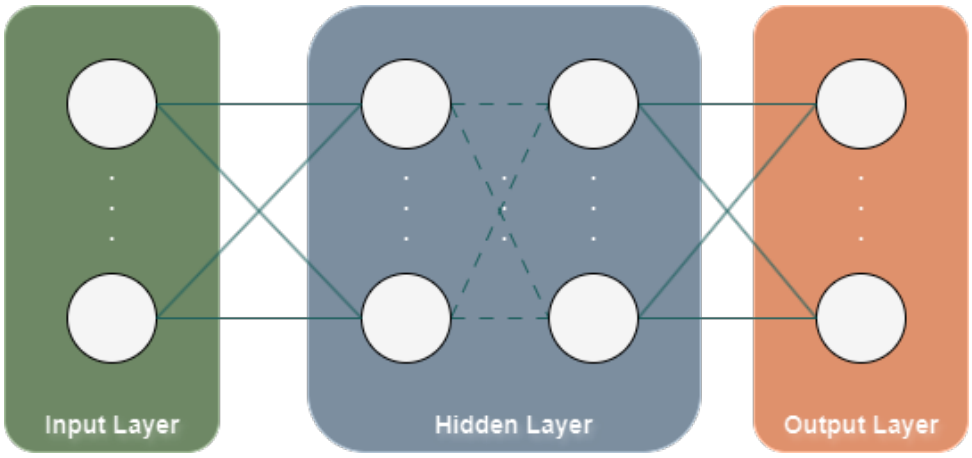

Figure 3.17. Multi Dense Model Visualization

3.5.2. Feature Interaction Network

In this network model type, the dense layers with the ReLU activation function are used to capture potential interactions between different bits of the RSA variables. This allows the network to learn more complex relationships than a simple linear model.

```
input_layer = Input(shape=(i_size,))

# Feature interaction layers
interaction = Dense(i_size // 2,
            activation='relu')(input_layer)
interaction = Dense(i_size // 4,
            activation='relu')(interaction)

output = Dense(o_size,
            activation='sigmoid',
            name='output')(interaction)
```

Figure 3.18. Feature Interaction Neural Network Model Code

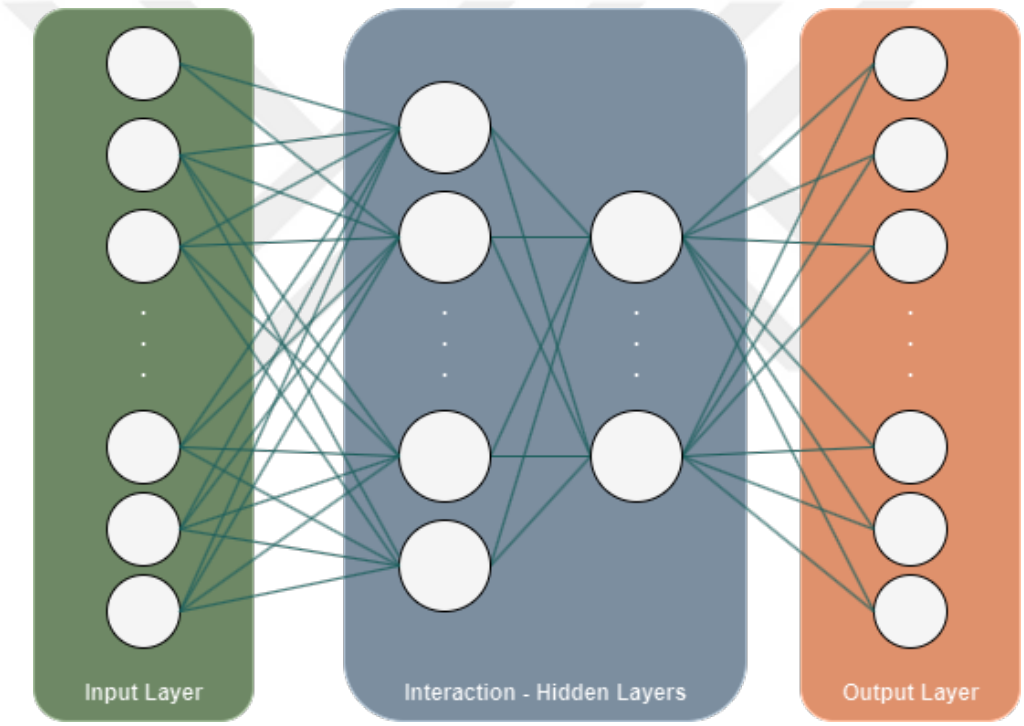

Figure 3.19. Feature Interaction Model Visualization

3.5.3. Autoencoder Feature Extraction Neural Network

The main reason for creating a model like this is because an autoencoder can be used to reduce unnecessary features affecting the outcome by reducing the dimensionality of the input (e.g., the semi-prime n) while retaining the most critical features. This could potentially help in identifying patterns or characteristics of the input data that relate to the factors (p and q).

The neuron connections of this neural network can be visualized like this:

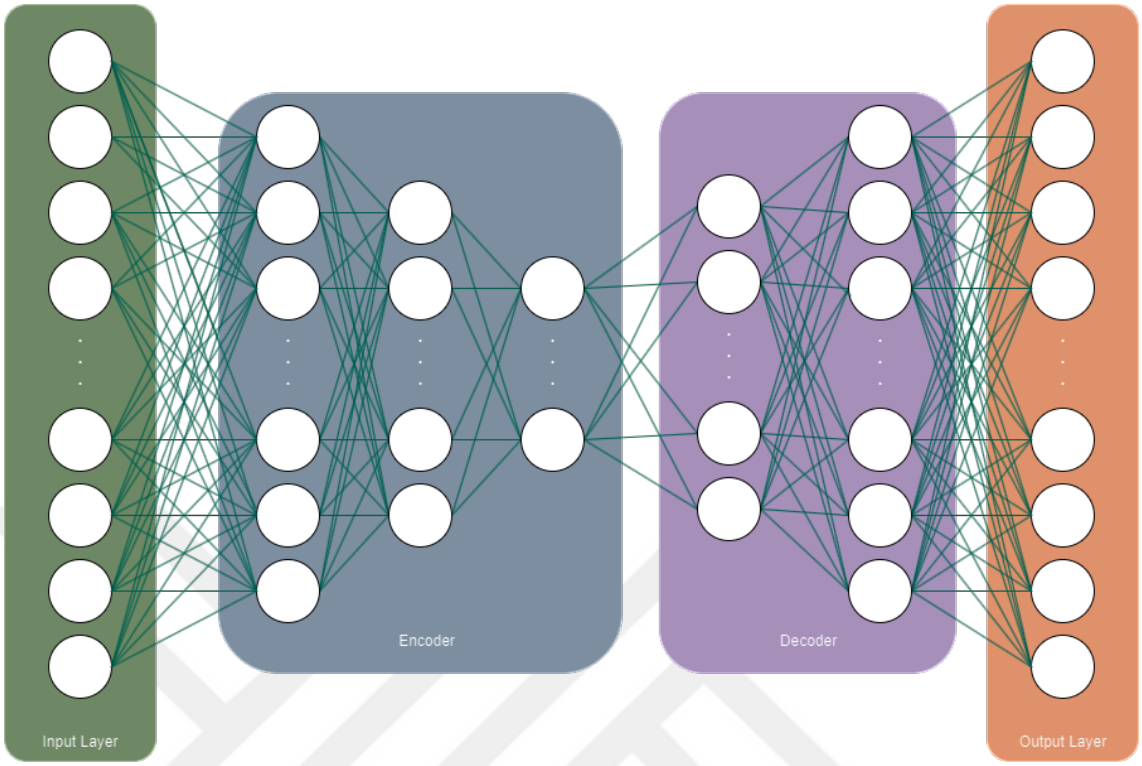

Figure 3.20. Autoencoder Feature Extraction Model Visualization

The code of this neural network model is shared below.

```
# Encoder
input_layer = Input(shape=(i_size,))
encoded = Dense(i_size//2,
                activation='relu')(input_layer)
encoded = Dense(i_size//4,
                activation='relu')(encoded)
encoded_output = Dense(i_size//8,
                activation='relu')(encoded)

# Decoder for predicting p and q
decoded_p = Dense(i_size//4,
                activation='relu')(encoded_output)
decoded_p = Dense(i_size//2,
                activation='relu')(decoded_p)
output = Dense(o_size, activation='sigmoid',
                name='output')(decoded_p)
```

Figure 3.21. Autoencoder Feature Extraction Neural Network Code

3.5.4. Hierarchical Neural Network

This neural network has two stages which are "**Intermediate Prediction**" and "**Final Prediction**".

In the intermediate prediction stage, the input layer passes its nodes through two dense layers. The first layer has half the nodes of the input layer and the second has quarter nodes of the input layer, both using the ReLU activation function. This stage's output is a single intermediate value, produced by a layer with one node and a linear activation function.

In the final prediction stage, the intermediate value is combined with the original input through concatenation. This new input passes through two more dense layers with the same node sizes and ReLU activations. The final layer has the total number of output-size nodes with a sigmoid activation function, making predictions likely between 0 and 1.

This network structure first estimates an intermediate value and then refines the final prediction using both the intermediate value and the original input.

```
# First stage: Predict an intermediate value
input_layer = Input(shape=(i_size,))
intermediate = Dense(i_size // 2,
            activation='relu')(input_layer)
intermediate = Dense(i_size // 4,
            activation='relu')(intermediate)
intermediate_output = Dense(1, activation='linear',
    name='intermediate_output')(intermediate)

# Second stage: Predict p and q using
# the intermediate value
second_input = Concatenate()([input_layer,
                intermediate_output])
hidden = Dense(i_size // 2,
        activation='relu')(second_input)
hidden = Dense(i_size // 4,
        activation='relu')(hidden)
output = Dense(o_size, activation='sigmoid',
        name='output')(hidden)
```

Figure 3.22. Hierarchical Neural Network Code

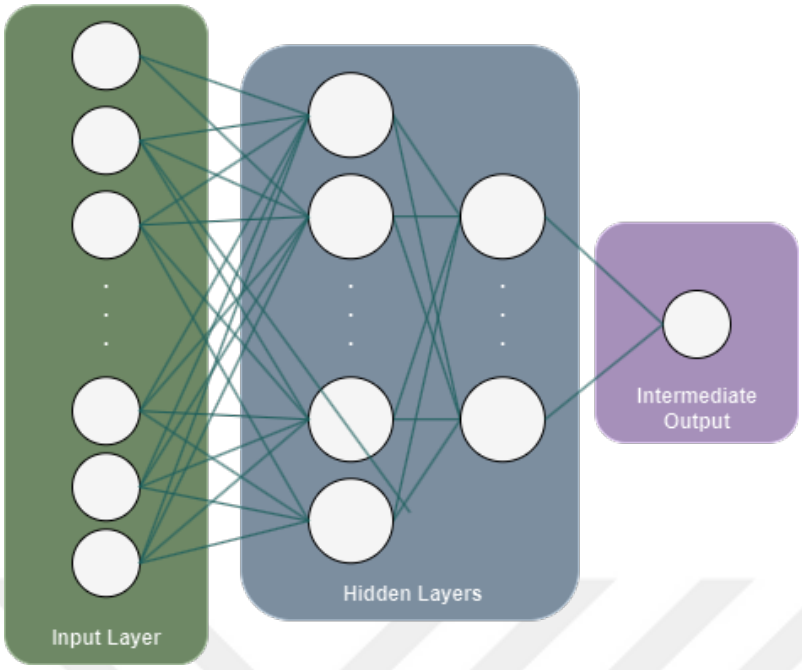

Figure 3.23. Hierarchical Model Visualization - First Stage

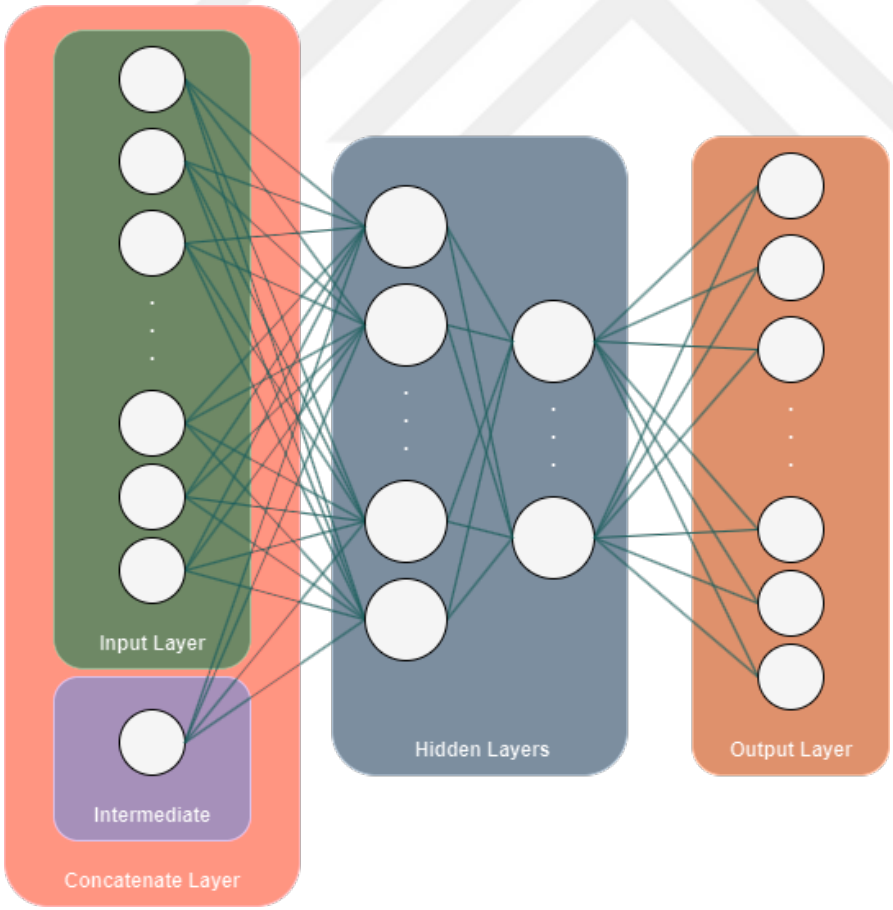

Figure 3.24. Hierarchical Model Visualization - Second Stage

3.5.5. Convolutional Feed Forward Neural Network

This neural network model consists of several layers designed to process data:

- **Convolutional Layer (Conv1D)**: This layer applies filters to the input data to capture patterns in sequences. It scans the data in small segments (3 units at a time) and outputs a set of features.
- **Flatten Layer**: The output from the convolutional layer is reshaped into a flat array, allowing it to be processed by the following dense layers.
- **Dense Layers**: These fully connected layers learn relationships between the features. The first and second dense layers use the ReLU activation function to enhance learning, while the final dense layer uses the softmax activation function to produce a probability distribution over the output classes.
- **Dropout Layer**: This layer randomly drops 20% of the connections during training to prevent overfitting, improving the model's generalization ability.

This combination of layers helps the model learn from sequential data and make predictions based on the extracted patterns.

```
self.model = Sequential([
    Conv1D(i_size, 3,
           activation='relu',
           input_shape=(i_size, 1)),
    Flatten(),
    Dense(i_size, activation='relu'),
    Dropout(0.2),
    Dense(i_size, activation='relu'),
    Dense(o_size, activation='softmax')
])
```

Figure 3.25. Convolutional Feed Forward Neural Network Code

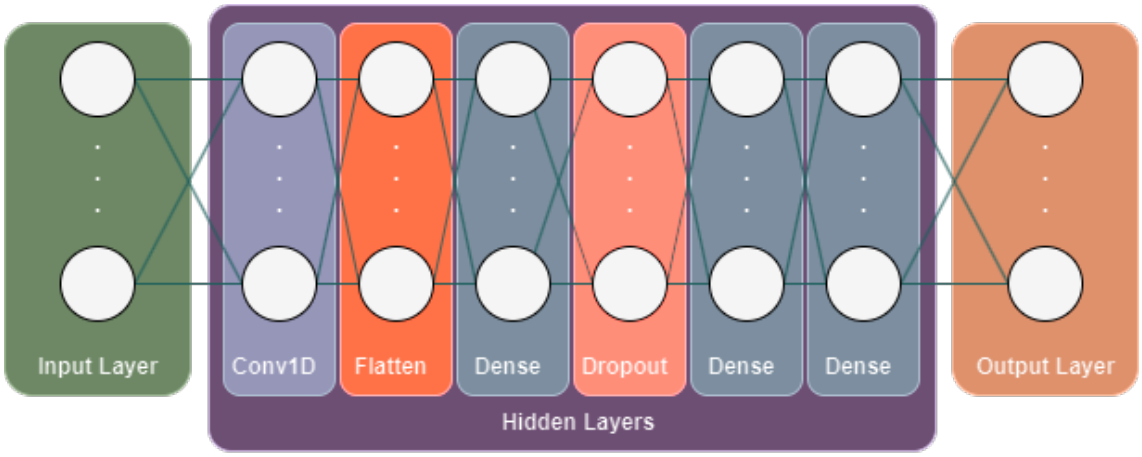

Figure 3.26. Convolutional Feed Forward Model Visualization

3.5.6. Hybrid Convolutional LSTM Neural Network

Even if it looks similar to the previous model, there are some differences between the previous model and this one. In this model, some layer types are changed with a MaxPooling1D layer and an LSTM layer. Also, some new parameters are added or old ones are updated in layers like Conv1D and Dropout. The code and the visualizations of the model are shared below.

```
self.model = Sequential()

self.model.add(Conv1D(filters=64, kernel_size=3,
                      padding='same',
                      activation='relu',
                      input_shape=(i_size, 1)))
self.model.add(MaxPooling1D(pool_size=2))
self.model.add(LSTM(i_size//4, activation='tanh',
                    return_sequences=True))
self.model.add(Dropout(0.3))
self.model.add(Flatten())
self.model.add(Dense(o_size, activation='sigmoid'))
```

Figure 3.27. Hybrid Convolutional LSTM Neural Network Code

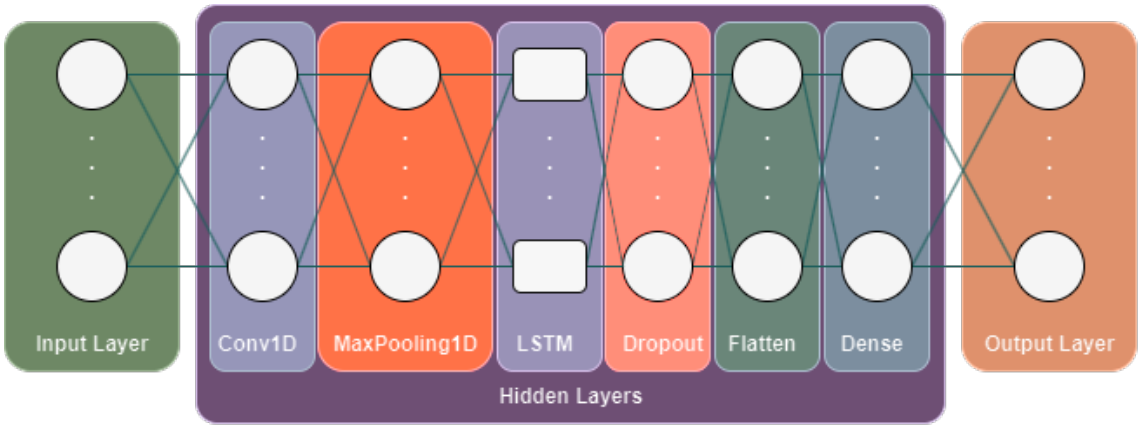

Figure 3.28. Hybrid Convolutional LSTM Model Visualization

3.5.7. Conv2D Neural Network

This one is the last model tested in this project. In this one, the input and output data are considered as matrices. And the model is trained with Conv2D layers. The code and the visualization of the model are shared below.

```
self.model = Sequential()

self.model.add(Conv2D(4, i_size,
                      activation='relu',
                      padding='same'))
self.model.add(Conv2D(2, i_size,
                      activation='relu',
                      padding='same'))
self.model.add(Conv2D(2, i_size,
                      activation='relu',
                      padding='same'))
self.model.add(Conv2D(1, o_size,
                      activation='sigmoid',
                      padding='same'))
```

Figure 3.29. Conv2D Neural Network Code

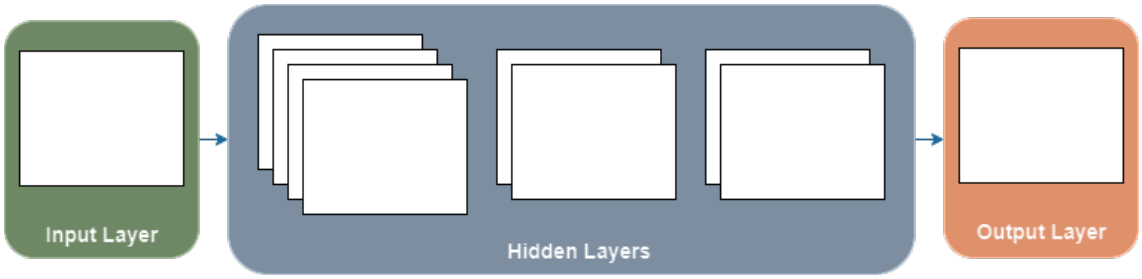

Figure 3.30. Conv2D Model Visualization

After the initialization of these network topologies, determining the most suitable topology among them involves selecting the input and output data for training. The variables used to create input and output data are converted into binary format to facilitate the binary approach in the models. Additionally, increasing the value of epochs will likely enhance the accuracy of some models. However, when deciding to increase the epoch value, adjusting the epoch values for all models is essential to maintain consistency. Consequently, this may result in a more time-consuming training process. Because of that, selecting a reasonably low value for the epoch would be a better choice to reduce the training time and create more comparable models.

4. RESULTS AND DISCUSSION

In this section, the results of the conducted experiments are shared.

4.1. One Dimensional Model Results

The results of the trained neural networks are categorized into three distinct groups. Each group corresponds to the bit lengths of the prime numbers used as the training data. The training process of the largest bit-length dataset is not performed due to the time consumption of the training process and the predictability of the outcome based on the results of the models trained with the smaller bit-length datasets.

For each data model, the accuracies of the trained topology models obtained over 100 epochs for each model, are shared in the tables below. Each bit-length group includes two tables. Combining the accuracy metrics of all topologies in one table may make the table a bit crowded. However, this way is more suitable for comparison purposes. The names of the topologies are shortened due to page width constraints.

This detailed breakdown facilitates a comprehensive understanding of the model's performance across different training conditions.

4.1.1. Topology Abbreviations

- **AFE** -> Autoencoder Feature Extraction
- **CFF** -> Convolutional Feature Fusion
- **FIN** -> Feature Interaction Network
- **HN** -> Hierarchical Network
- **HCL** -> Hybrid Convolutional LSTM
- **MD** -> Multi Dense

The accuracy tables of these topologies are listed below.

Table 4.3. 256-bit Prime Dataset Model Accuracies

Dataset Model	**Acc. (AFE)**	**Acc. (CFF)**	**Acc. (FIN)**	**Acc. (HN)**	**Acc. (HCL)**	**Acc. (MD)**
Model-n-pq	.5054	.4983	.5047	.5041	.5043	.505
Model-n-p	.5061	.5002	.5052	.5055	.5058	.5054
Model-n-q	.5039	.4964	.5028	.5035	.5031	.5034
Model-n-phi	.5557	.5068	.5655	.5515	.5065	.5589
Model-s-p	.505	.5001	.5053	.5053	.5047	.5054
Model-s-q	.5039	.4964	.5044	.5035	.5042	.504
Model-se-p	.5055	.5001	.5049	.5054	.5057	.505
Model-so-p	.5058	.5001	.5048	.5047	.5059	.5056
Model-so-a	.5066	.5068	.5066	.5067	.5065	.5066

Table 4.4. 512-bit Prime Dataset Model Accuracies

Dataset Model	**Acc. (AFE)**	**Acc. (CFF)**	**Acc. (FIN)**	**Acc. (HN)**	**Acc. (HCL)**	**Acc. (MD)**
Model-n-pq	.5021	-	.5021	.5021	.5016	.5017
Model-n-p	.5026	-	.5017	.5028	-	.502
Model-n-q	.502	-	.5015	.5021	-	.5009
Model-n-phi	.5632	-	.5305	.5271	-	.5234
Model-s-p	.5029	.5003	.5032	.5026	-	.5034
Model-s-q	.5029	.4983	.502	.5008	.502	.5024
Model-se-p	.5034	-	.5023	.5014	-	.5032
Model-so-p	.5019	.5002	.5036	.5027	-	.503
Model-so-a	.503	.5029	.5028	.5034	-	.503

Table 4.5. 1024-bit Prime Dataset Model Accuracies

Dataset Model	Acc. (AFE)	Acc. (CFF)	Acc. (FIN)	Acc. (HN)	Acc. (HCL)	Acc. (MD)
Model-n-pq	.501	-	.501	.5008	-	.5009
Model-n-p	.5013	-	.5006	.501	-	.5009
Model-n-q	.5008	-	.5013	.5005	-	.5004
Modeln-phi	.5017	-	.501	.5013	-	.5012
Model-s-p	.5016	-	.5014	.5007	-	.5018
Model-s-q	.5015	-	.5005	.5007	-	.5006
Model-se-p	.5017	-	.5007	.5011	-	.5012
Model-so-p	.5017	-	.5016	.5008	-	.5018
Model-so-a	.5051	-	.5049	.5052	-	.5048

4.2. Two Dimensional Model Results

The accuracy results of the Conv2D Neural Network for 3 different bit lengths are shared in the table below.

Table 4.6. The Accuracies of the Conv2D Topology Model

Dataset Model	Acc. (256)	Acc. (512)	Acc. (1024)
Model-m-n-pq	.5049	.5005	.5003

4.3. Bit Position Errors

At the end of the training and testing processes, the false predictions were carefully examined by counting the positions where the models were predicted wrong. Since the accuracy metrics of all models are very similar, the bit position error graphs are also very similar. So, the results of some models that have different graphs are presented below to offer a clear picture of the challenges faced. This analysis shows all bit positions where the models had trouble, providing valuable insights into their performance and suggesting areas for further improvement.

The following graphs (in this section) display the bit position errors of models trained with 256-bit data groups.

The first graph shows the results of a model trained with the ModelNPQ data configuration and Autoencoder Feature Extraction topology. As can be seen, almost all bit positions are mispredicted at a rate of 50%. The graph of most models trained with ModelNPQ data configuration looks similar to this one.

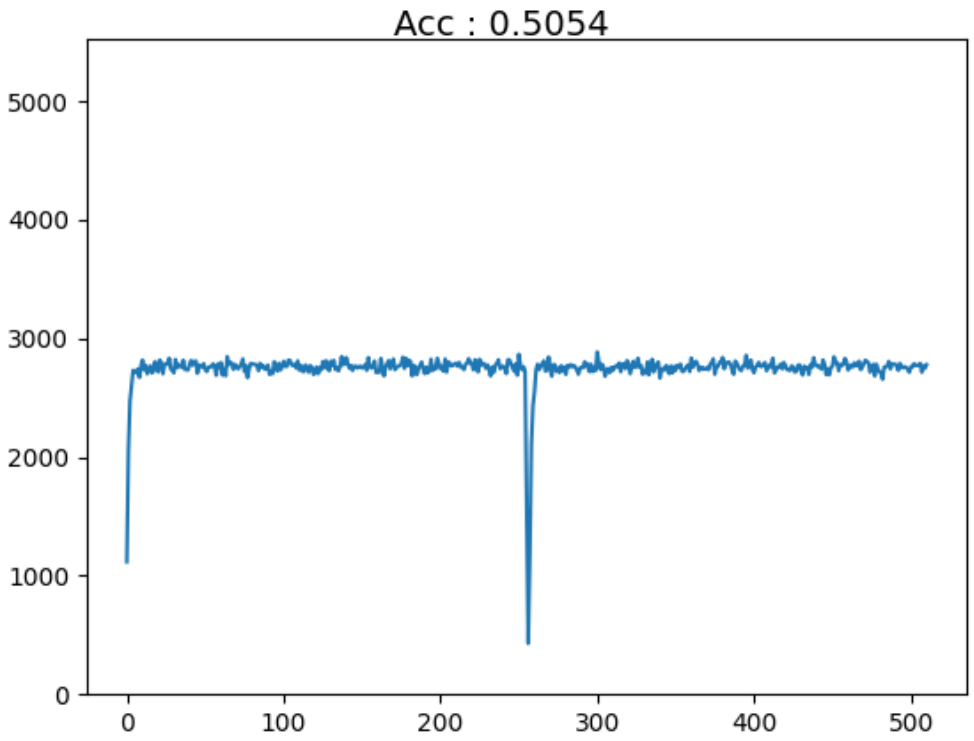

Figure 4.31. Autoencoder Feature Extraction - ModelNPQ - Bit Error Graph

For the other trained neural networks, most of the graphs resemble the one for the model trained with the ModelNP configuration using the Autoencoder Feature Extraction topology. Therefore, only this graph is shared to illustrate what the general bit position error graph looks like.

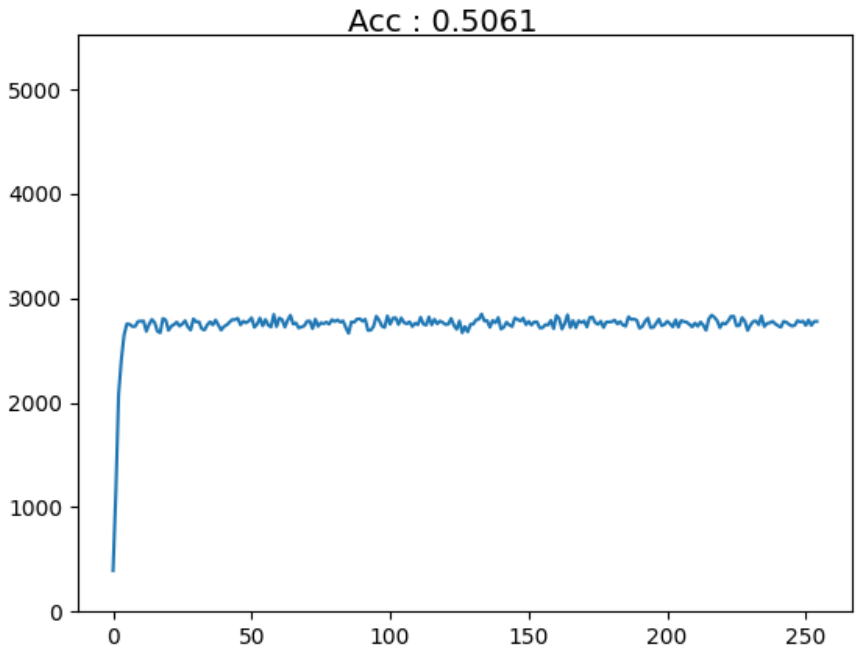

Figure 4.32. Autoencoder Feature Extraction - ModelNP - Bit Error Graph

The next belongs to the model trained with ModelNPHI using Hierarchical topology.

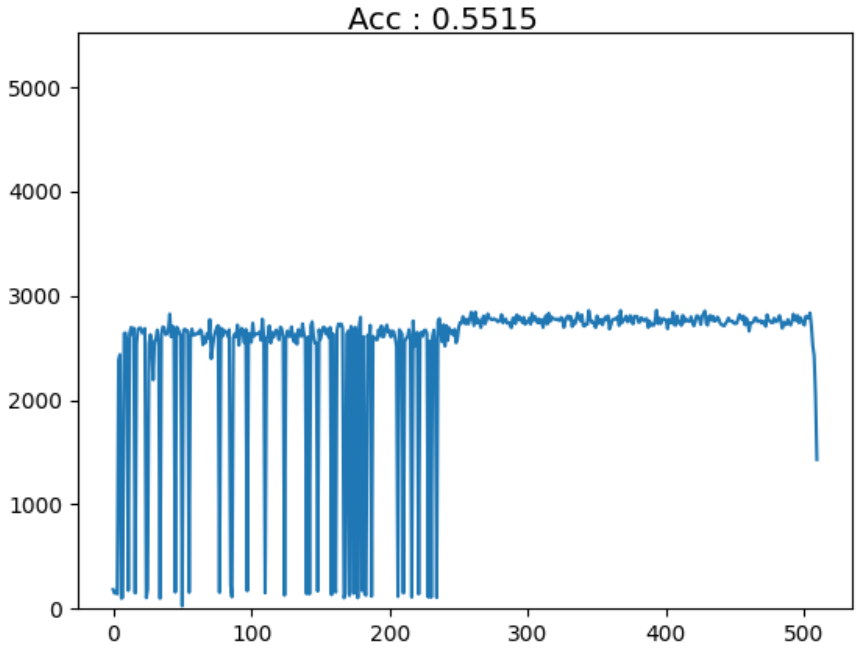

Figure 4.33. Hierarchical - ModelNPHI - Bit Error Graph

Finally, the last one is from the model trained with ModelNPHI using MultiDense topology.

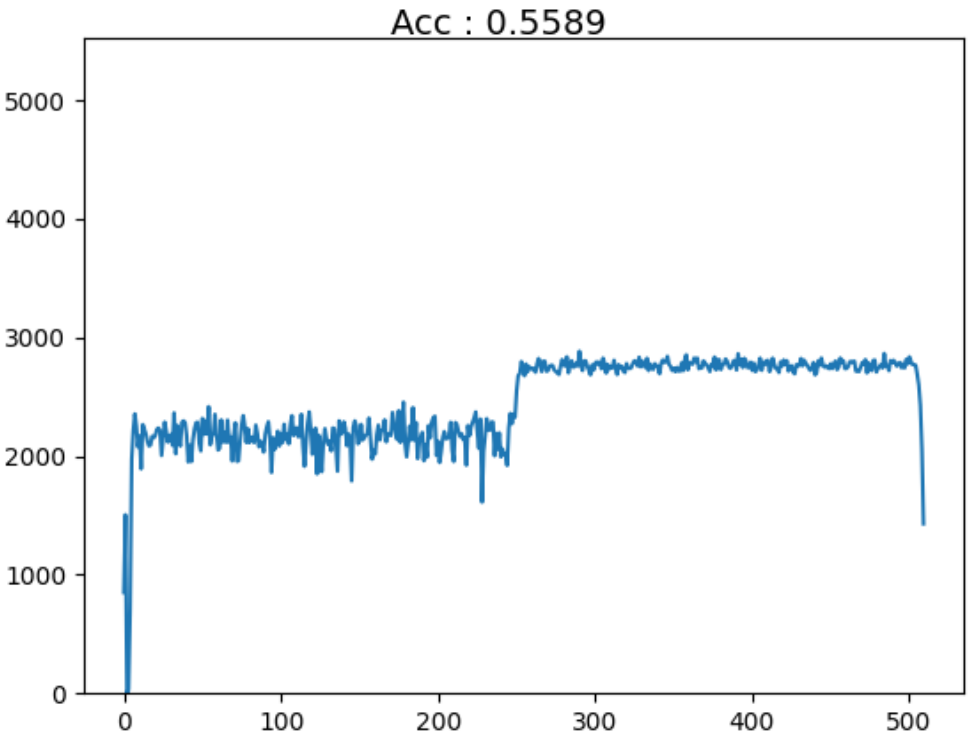

Figure 4.34. MultiDense - ModelNPHI - Bit Error Graph

At first glance, some bit positions in the first half of the bit positions for the Model-NPHI data could be useful in creating patterns. However, it is because the first half bits of the value of phi are very similar to the first half bits of the value of n.

4.4. Low Bit Length Testing

At the end of the training process using large bit-length datasets, smaller bit-lengths with larger datasets were tested to evaluate the effectiveness of the models used in this project. The first test involved training a model on a dataset containing 64-bit prime numbers with over 500,000 rows, using 1000 epochs. The training took approximately 15 hours, but the model's accuracy remained around 55%, showing minimal improvement. Following this, another dataset was created with 32-bit prime numbers, also with about

500,000 rows, and trained for 1000 epochs. In this case, training took around 8 hours per model, and the accuracy of some models reached approximately 70%.

The graphs below show that when the bit length increases, the training quality drops, and predictions for the bit positions will be reduced. To counter that, the amount of data used for training and also training parameters such as learning rate, epoch, etc. need to be changed to make the training process longer. The graphs are listed from the lowest bit length group to the highest bit length group.

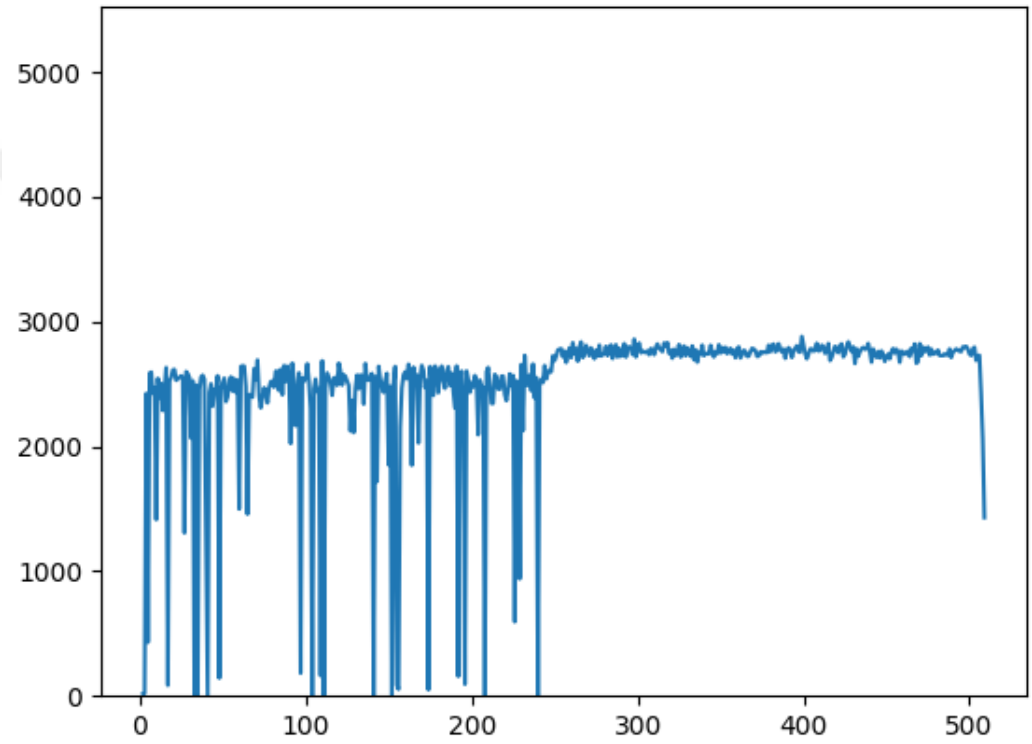

Figure 4.35. ModelNPHI - 256 Bit Length Group - Bit Error Graph

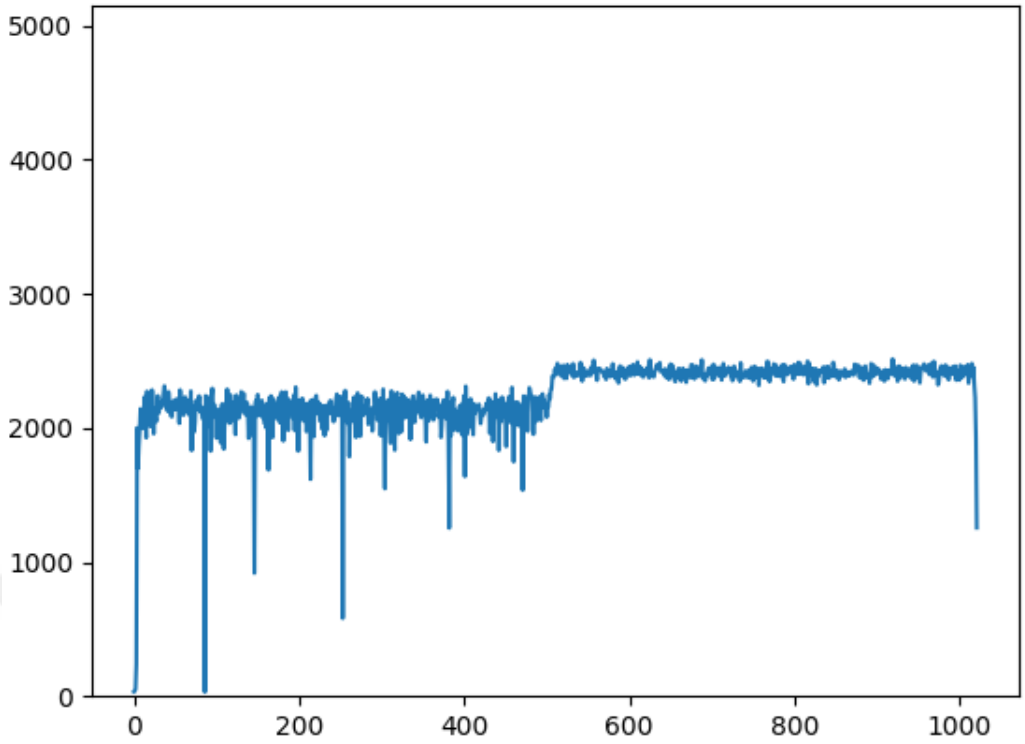

Figure 4.36. ModelNPHI - 512 Bit Length Group - Bit Error Graph

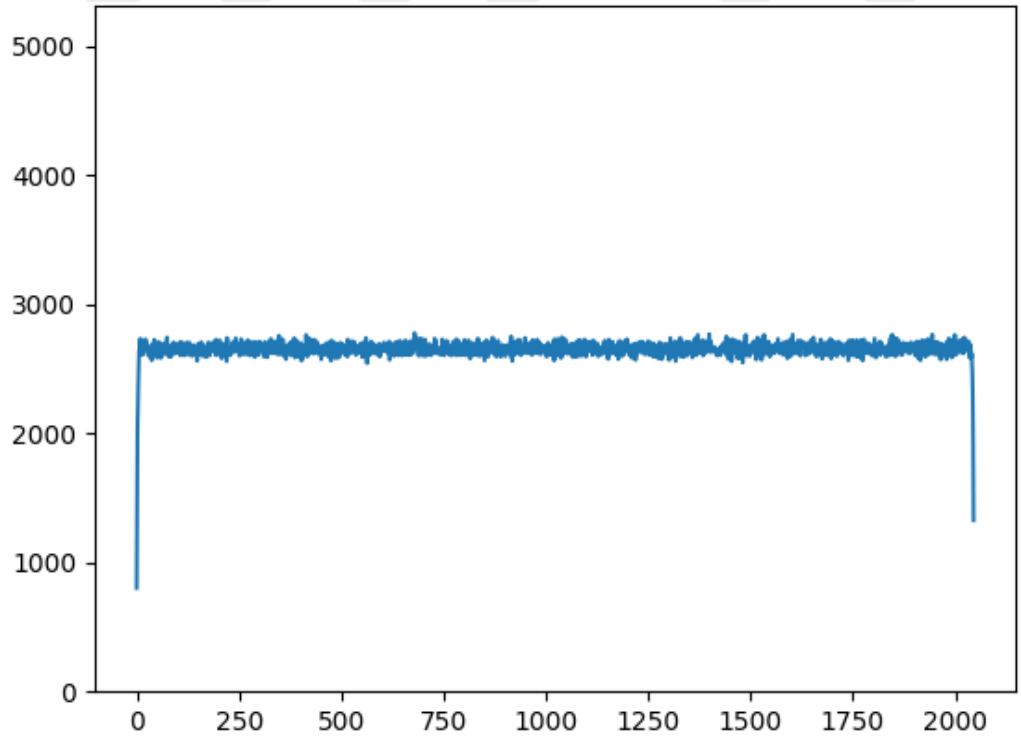

Figure 4.37. ModelNPHI - 1024 Bit Length Group - Bit Error Graph

The chart below presents the success rates of various models based on their bit lengths. It is clear that, for most models trained under the same conditions, success rates decrease as the bit length increases.

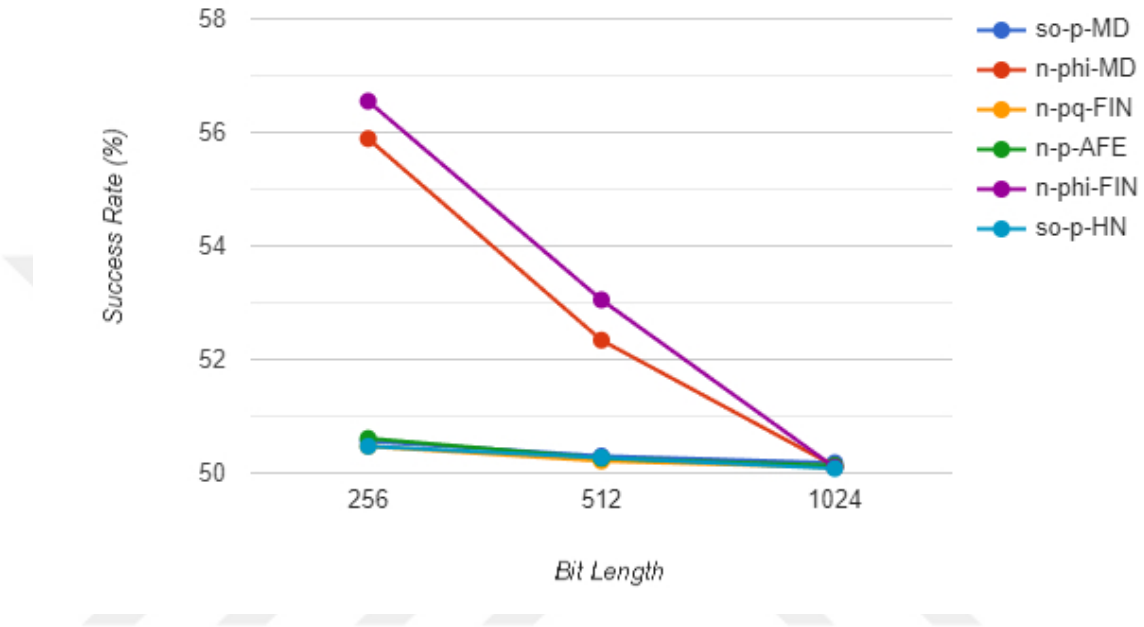

Figure 4.38. Bit Length - Success Rate Chart

5. CONCLUSION

In conclusion, ten different input-output pairs across three-bit lengths are trained with seven different neural network topologies in this thesis. Nine of these ten datasets are tested on six of these seven neural network models and "Model-m-n-pq" is designed to be used on the Conv2D neural network model. So, a total of 165 models are trained and tested in this thesis. Despite efforts to calibrate and assess the predictive capabilities of the developed models, the results show that the models were ineffective in accurately predicting the expected outputs. This highlights the complexity of creating machine-learning models for factorizing semi-prime numbers and the significant challenge posed by the RSA encryption algorithm.

Furthermore, an exhaustive examination of the frequency distribution of bit locations yielded no discernible patterns or significant discoveries for the predictions of the trained models. This outcome underscores the intricate nature of the underlying factors governing the distribution of prime factors within semi-prime numbers, elucidating the formidable barrier to successful decryption.

However, additional testing on the low-bit-length dataset suggests that using a larger dataset and training with a higher epoch value may improve the accuracy of the model's bit predictions. Furthermore, as the bit length increases, the need for a custom neural network model designed to handle the input and output bits specifically for the job becomes more critical. Implementing a customized neural network for this problem is likely to improve prediction accuracy while also reducing the required training time for the model.

While the results of this investigation may not have met the initial expectations, they nonetheless contribute significant insights to the ongoing discussion about cryptanalysis methodologies. These findings underscore the need for continued research and innovation in developing machine-learning models tailored to cryptanalysis, intending to overcome the inherent challenges posed by complex encryption algorithms such as RSA.

Moving forward, avenues for further exploration could include refining and optimizing existing machine-learning algorithms and exploring novel methodologies that may offer greater efficacy in factorizing semi-prime numbers. Also, the importance of the training dataset size and allowing sufficient time for adjusting the connections between

nodes for the given data should not be overlooked. Additionally, future research endeavors could benefit from a multidisciplinary approach, drawing upon insights from fields such as mathematics, computer science, and artificial intelligence to address the intricate challenges posed by cryptographic systems. Through sustained efforts and collaboration, significant progress is expected in advancing the state-of-the-art in cryptanalysis, which will in turn strengthen the security and integrity of digital communication systems.

6. REFERENCES

Aboud, S. J., 2009. Efficient Method for Breaking RSA Scheme. *Ubiquitous Computing and Communication Journal*, NTMS, no. 1:1–5.

Abubakar, A., Muhammad, S., Tijjani, B., Zeki, A., Chiroma, H., Usman, M., Raji, S. and Mahmud, M., 2014. Cryptanalytic Attacks on Rivest, Shamir, and Adleman (RSA) Cryptosystem: Issues and Challenges. *Journal of Theoretical and Applied Information Technology*, 61:1–7.

Aggarwal, D. and Maurer, U., 2009. Breaking RSA Generically is Equivalent to Factoring. *EUROCRYPT 2009*, pp 1–18.

Alrasheed, A. and Fatima. RSA Attacks, 2021. Course paper for CPSC 4600/5600 (Biometrics and Cryptography), University of Tennessee at Chattanooga.

Arora, B., Srishti, K., Khatri, N. and Niranjan, V., 2015. Application of Artificial Neural Network in Cryptography. *2nd International Conference on Power Energy, Environment and Intelligent Control*, pp 18–19.

Blake, S., 2023. Integer Factorisation, Fermat & Machine Learning on a Classical Computer. *arXiv:2308.12290 [cs.LG]*.

Borders, W., Pervaiz, A., Fukami, S., Camsari, K., Ohno, H. and Datta, S., 2019. Integer factorization using stochastic magnetic tunnel junctions. pp 390–393.

Brown, D. R. L., 2016. Breaking RSA May Be As Difficult As Factoring. *Journal of Cryptology*, 20:220–241.

Brown, S. Machine learning, explained, 2021. Accessed: 2023-10-09.

Dass, P., Sharma, H., Bansal, J. and Nygard, K., 2013. Meta heuristics for prime factorization problem. *World Congress on Nature and Biologically Inspired Computing*, pp 126–131.

Jansen, B. and Nakayama, K., 2005. Neural Networks Following a Binary Approach Applied to the Integer Prime-Factorization Problem. *Proceedings. 2005 IEEE International Joint Conference on Neural Networks*, pp 2577–2582.

Jiang, S., Britt, K., McCaskey, A., Humble, T. and Kais, S., 2018. Quantum Annealing for Prime Factorization. *Scientific Reports*, 8:17667.

Kocher, P., 1996. Timing attacks on implementations of Die-Hellman, RSA, DSS, and other systems. *CRYPTO 96*, 1109:104–113.

Lehman, R. S., 1974. Factoring Large Integers. *Mathematics of Computation*, 28:637–646.

Meletiou, G., Tasoulis, D. and Vrahatis, M., 2002. A first study of the neural network approach to the RSA cryptosystem. *Sixth IASTED International Conference on Artificial Intelligence and Soft Computing (ASC 2002)*, pp 483–488.

Miller, G. and Rieman, 1976. Hypothesis and Tests for Primality. *Journal of Computer and System Sciences*, pp 300–317.

Mishra, M., Chaturvedi, U. and Pal., S., 2014. A multithreaded bound varying chaotic firefly algorithm for prime factorization. *2014 IEEE International Advance Computing Conference (IACC)*, pp 1322–1325.

Mishra, M., Chaturvedi, U. and Shukla, K., 2016. Heuristic algorithm based on molecules optimizing their geometry in a crystal to solve the problem of integer factorization. *Soft Computing*, 20(9):3363–3371.

Monaco, J. V. and Vindiola, M. M., 2017. Integer factorization with a neuromorphic sieve. *IEEE International Symposium on Circuits and Systems (ISCAS)*, pp 1–4.

Murat, B., Kadyrov, S. and Tabarek, R., 2021. Integer Prime Factorization with Deep Learning. *Suleyman Demiral University Journal*, pp 1–5.

Pollard, J., 1975. A Monte Carlo Method For Factorization. *BIT*, 15:331–334.

Pomerance, C., 1984. The Quadratic Sieve Factoring Algorithm. *EUROCRYPT 1984*, 209:169–182.

Portigliatti, T., Maghrebi, H. and Prouff, E., 2016. Breaking Cryptographic Implementations Using Deep Learning Techniques. *International Conference on Security, Privacy, and Applied Cryptography Engineering*, pp 3–26.

Rutkowski, E. and Houghten, S., 2020. Cryptanalysis of RSA: integer prime factorization using genetic algorithms. *2020 IEEE Congress on Evolutionary Computation*, pp 1–8.

Saif, A. and Abidi, M. R., 2019. Machine Learning Based Attack on Certain Encryption Schemes. *2nd International Conference on Computer Applications & Information Security (ICCAIS)*, pp 1–6.

Sangouard, N. and Gouzien, E., 2021. Factoring 2048-bit RSA Integers in 177 Days with 13436 Qubits and a Multimode Memory. *Phys. Rev. Lett. 127, 140503*, 2:1–5.

Seker, S. E. and Mert, C., 2013. Reverse Factorization and Comparison of Factorization Algorithms in attack to RSA. *Proceedings of the International Conference on Scientific Computing*, 1:1–5.

Sherstinsky, A., 2020. Fundamentals of Recurrent Neural Network (RNN) and Long Short-Term Memory (LSTM) Network. *Physica D: Nonlinear Phenomena*, 10:1–43.

Smiljanić, J. M. and Ivaniš, P. N., 2011. Attacks on the RSA cryptosystem using integer factorization. *2011 19thTelecommunications Forum (TELFOR) Proceedings of Papers*, pp 550–553.

Somsuk, K., 2014. A new modified integer factorization algorithm using integer modulo 20's technique. *2014 International Computer Science and Engineering Conference (ICSEC)*, 10(2):1469 1476.

Son, K. K. and Rasool, A., 2018. Cryptographic Attack Possibilities over RSA Algorithm through Classical and Quantum Computation. *International Conference on Smart Systems and Inventive Technology (ICSSIT)*, pp 11–15.

Villegas, F. I. L. and Cordero, C. V., 2021. Machine Learning Analysis for Side-Channel Attacks over Elliptic Curve Cryptography. *IEEE CHILEAN Conference on Electrical, Electronics Engineering, Information and Communication Technologies (CHILECON)*, pp 1–20.

Vostrov, G. and Dermenzhy, I., 2019. The Concept of Machine Learning and Elliptic Curves United Approach in Solving of the Factorization Problem. *XIth International Scientific and Practical Conference on Electronics and Information Technologies (ELIT)*, pp 87–91.

Yampolskiy, R., 2010. Application of the bio-inspired algorithm to the problem of integer factorisation. *International Journal of Bio-Inspired Computation*, 2(2):115–123.

CURRICULUM VITAE

Ahmet GÜRDAL

EDUCATION

Master of Sciences (2022 - 2024)	Akdeniz University Institute of Natural and Applied Sciences Department of Computer Engineering, Antalya
Bachelor of Science (2016 - 2020)	Akdeniz University Faculty of Engineering Department of Computer Engineering, Antalya

WORK EXPERIENCE

Full Stack Developer (2022 - Current)	Revolvia (Media Technology Company)
Software Developer (2021 - 2022)	6Zeka (Finance Technology Software Company)
System Support and Data Engineer (2017 - 2020)	Devexperts (Finance Technology Software Company)

PUBLICATIONS