TRAINING PROJECT REPORT
ON

CRYPTANALYSIS USING MACHINE LEARNING
Differentiating Real and Random Ciphertext Pairs

Submitted by
PARISHA KHURANA

NETAJI SUBHAS UNIVERSITY OF TECHNOLOGY, DWARKA

Under the Guidance of

DR. RAJESH ASTHANA
Scientist- ‘F’

Scientific Analysis Group
Defence Research & Development Organisation

In fulfillment of the completion of the training program July 2024

CERTIFICATE

This is to certify that this Report Titled “**CRYPTANALYSIS USING MACHINE LEARNING, Differentiating Real and Random Ciphertext Pairs**” embodies work done by Parisha Khurana, B.Tech in Computer Science and Engineering(with Specialisation in Artificial Intelligence), Netaji Shubas University of Technology, Delhi, at Scientific Analysis Group, Metcalfe House, Delhi, 110054. He carried out this work during the period of 20th May 2024 to 31st July 2024 under the supervision of Dr. Rajesh Asthana, SAG, DRDO.

Mr. Rajesh Asthana

(Scientist ‘D’, SAG, DRDO)

Abstract

This report is focused on the application of machine learning techniques to enhance cryptanalysis, particularly for lightweight encryption ciphers like AES (Advanced Encryption Standard) and XOR-based encryption. Traditional differential cryptanalysis, which relies on analyzing differences between pairs of plaintexts and their corresponding ciphertexts to uncover cryptographic keys, is often limited by the complexity and randomness of modern encryption algorithms.

The core idea of this research is to use machine learning models to differentiate between "real" ciphertext pairs (generated by encrypting meaningful plaintexts) and "random" ciphertext pairs (without meaningful correlation). The machine learning model is trained on features extracted from these ciphertext pairs, enabling it to identify patterns that traditional methods might miss.

This approach leverages the ability of machine learning to handle complex relationships within the cipher and potentially find more effective distinguishers with lower computational effort. The ultimate goal is to improve the efficiency and effectiveness of cryptanalysis by using machine learning as a tool to reveal vulnerabilities in encryption algorithms.

Acknowledgments

I thank all personalities responsible behind the success of this project, especially the following ones.

I thank **Smt U. Jeya Santhi** Director, Scientific Analysis Group (SAG), Defence Research and Development Organisation (DRDO), for allowing me to conduct this project and support it. A special gratitude I give to my mentor **Dr. RAJESH ASTHANA,** Scientist ‘F’, SAG, DRDO, whose contribution in stimulating suggestions and encouragement, helped me to coordinate my project especially in implementing this project. Without his wise counsel and able guidance, it would have been impossible to complete the project in this manner. I am indebted to him for his kind encouragement and in-depth knowledge about the subject, which helped me in overcoming the difficulties that arose in my project.

I would like to thank the principal of **Dr. MPS Bhatia,** Head of Training and Placement Cell, for providing this opportunity to carry out this project in two months of industrial training. The constant guidance and encouragement received from **Dr. Bijendra Kumar,** H.O.D. CSE Department, Netaji Subhas University of Technology, Dwarka has been a great help in carrying out the project work and is acknowledged with reverential thanks.

I appreciate the guidance given by other supervisors as well as the panels especially in our project presentation that has improved our presentation skills thanks to their comments and advice.

Parisha Khurana

College Roll No. 2022UCA1891

Department Computer Science & Engineering

Netaji Subhas University of Technology (NSUT)

Dwarka, Delhi

Declaration

I hereby declare that this project entitled "**Non-Linear Mapping for Encrypted Datasets**" being submitted to the Department of Computer Science & Engineering of Netaji Subhas University of Technology (NSUT Dwarka, Delhi) for the fulfillment of the completion of training program in June 2024, is a bonafide record of original work carried out under the guidance and supervision of Dr. Rajesh Asthana, Scientist F, SAG, DRDO and has not been presented elsewhere.

Parisha Khurana
Roll No. 2022UCA1891

Contents

Chapter 1:

Introduction to Organisation

1.1 Defence Research and Development Organisation

The Defence Research and Development Organisation (DRDO) is a premier agency of the Government of India, responsible for military research and development. Headquartered in New Delhi, DRDO was established in 1958 through the merger of the Technical Development Establishment and the Directorate of Technical Development and Production of the Indian Ordnance Factories with the Defence Science Organization. Operating under the administrative control of the Ministry of Defence, DRDO has grown to become India's largest and most diverse research organization. DRDO's vast network includes over 50 laboratories, each focused on developing cutting-edge defense technologies across various domains such as aeronautics, armaments, electronics, combat vehicles, engineering systems, missiles, advanced computing, simulation, special 12 materials, naval systems, life sciences, and information systems. With a team of approximately 5,000 scientists from the Defence Research & Development Service (DRDS) and about 25,000 other scientific, technical, and support personnel, DRDO is dedicated to enhancing India's self-reliance in defense systems. The organization's efforts are directed towards designing and developing world-class weapon systems and equipment to meet the qualitative requirements of the Indian Armed Forces. In addition to meeting the nation's defense needs, DRDO also contributes significantly to the broader society through spinoff technologies and innovations that support national development.

1.1.1 Scientific Analysis Group

The Scientific Analysis Group (SAG) is a specialized division within DRDO, focusing on the areas of cryptology and information security. Established in 1963, SAG was initially tasked with developing new scientific methods for the design and analysis of communication systems. Starting with just 12 scientists and based in the Central Secretariat Complex, the group has since expanded significantly, both in terms of manpower and facilities. By 1973, SAG was placed under the direct control of the Chief Controller (R&D) and became a full-fledged Directorate within DRDO's R&D Headquarters. In 1976, SAG began taking on research and development projects in mathematical analysis, communication, and speech analysis. As its responsibilities grew, SAG moved to the Metcalfe House Complex and was further strengthened with enhanced electronic facilities and an expanded workforce. Today, SAG operates from two independent buildings, with a total staff of 140 scientists, technical experts, and other personnel. The group's mission is to develop state-of-the-art tools and techniques in contemporary mathematics, computer science, and electronics & communication for the analysis of security and IT products. SAG also focuses on creating both generalized and domain-specific analytical algorithms and tools, as well as establishing advanced facilities for electronic probing, communication signal, and protocol analysis. 13 Scientific Analysis Group (SAG) is working in the area of cryptology and information security. SAG is developing tools and techniques based on contemporary mathematics, computer science and electronics, and communication for information security

1.1.1.1 Vision

To make the Scientific Analysis Group the finest Cryptology and Information Security Laboratory in the World.

1.1.1.2 Mission

- To develop tools and techniques based on contemporary Mathematics, Computer Science and
- Electronics & Communication for Analysis of Security and IT products.
- To build generalized as well as domain/system-specific analytical algorithms and tools.
- To establish state-of-the-art facilities for electronic probing, communication signal & protocol analysis

1.1.1.3 Area of Work

- Advanced Mathematical and Statistical Analysis & Development of Tools
- Linguistics - Computational and Structural

- Speech Analysis - Recognition and Synthesis
- Simulation Studies Microprocessor-based Systems
- Signal Processing
- Satellite Communication
- High Performance Computing

Chapter 2

Introduction

2.1. Overview of Cryptanalysis:

Cryptanalysis, the science of deciphering encrypted messages without access to the key, has been a cornerstone of cryptography for decades. The evolution of cryptanalysis mirrors the development of cryptographic systems themselves, as cryptanalysts have continuously developed new methods to uncover vulnerabilities and break cryptographic algorithms.

In an event or discussion, you can delve into how traditional cryptanalysis techniques like Differential Cryptanalysis and Linear Cryptanalysis have played crucial roles in the field, particularly in analyzing and attacking encryption algorithms.

Types of Cryptanalysis:

1. **Classical Cryptanalysis:**
    - Focuses on exploiting mathematical weaknesses in encryption algorithms.
    - Common techniques include frequency analysis, known plaintext attacks, and chosen plaintext attacks.
2. **Modern Cryptanalysis:**
    - Involves more advanced methods such as linear and differential cryptanalysis.
    - Uses statistical techniques to find correlations between plaintexts and ciphertexts that can lead to key recovery.

2.1.1 Traditional Techniques

1. **Differential Cryptanalysis:**

- **Historical Significance**: Differential Cryptanalysis is one of the most famous methods used in the history of cryptanalysis. It gained prominence when it was successfully applied to the Data Encryption Standard (DES), a widely used symmetric-key algorithm in the late 20th century. The technique was independently discovered by Eli Biham and Adi Shamir in the late 1980s, though it was later revealed that IBM and the NSA were aware of the method during DES's development.

- **How It Works**: The core idea behind Differential Cryptanalysis is to analyze how differences in plaintext pairs affect the differences in the resulting ciphertext pairs. By systematically comparing pairs of plaintexts that have specific differences and observing how these differences propagate through the encryption process, cryptanalysts can identify patterns or biases in the ciphertext. These patterns can be exploited to recover the secret key used in the encryption, especially if the algorithm exhibits non-random behavior.
- **Application to DES**: In the case of DES, Differential Cryptanalysis was able to significantly reduce the effort needed to break the cipher compared to a brute-force attack. While the DES was still considered secure for some time, this technique demonstrated that DES's 56-bit key length was insufficient for long-term security.

2. **Linear Cryptanalysis:**

- **Approximation with Linear Expressions**: Linear Cryptanalysis, introduced by Mitsuru Matsui in 1993, is another powerful technique used against block ciphers. This method involves finding linear approximations to the non-linear behavior of the cipher. Essentially, cryptanalysts create linear equations that approximate the relationships between plaintext, ciphertext, and the secret key.
- **Correlation with Ciphertext**: Once these linear approximations are established, they are used to analyze the actual ciphertext produced by the cipher. If the approximations hold true more often than expected by random chance, the cryptanalyst can use this correlation to gather information about the key. Over time, by analyzing many plaintext-ciphertext pairs, the key can be reconstructed with a high degree of accuracy.
- **Effectiveness**: Linear Cryptanalysis has proven effective against a variety of block ciphers, including DES. While DES withstood linear cryptanalysis better than differential cryptanalysis, the technique was still able to reduce the security margin of the cipher, further underscoring the need for stronger cryptographic designs.

Goals of Cryptanalysis:

- **Evaluate Security:** Determine the strength and weaknesses of cryptographic algorithms.
- **Enhance Encryption:** Improve existing cryptographic systems based on the identified vulnerabilities.
- **Develop Standards:** Establish guidelines and standards for secure cryptographic practices.

2.2 Challenges in Modern Cryptanalysis:

- **Resistance of Modern Algorithms**: Modern encryption algorithms, such as the Advanced Encryption Standard (AES), have been designed specifically to resist these traditional cryptanalysis methods. The designers of these algorithms have incorporated features to increase their statistical resistance, making it much harder for cryptanalysts to find exploitable patterns or approximations.
- **Computational Complexity**: One of the key challenges in modern cryptanalysis is the sheer computational complexity required to break contemporary ciphers. As ciphers have grown more sophisticated, the amount of computational power needed to successfully attack them has increased exponentially. This has led cryptanalysts to explore new methods, such as quantum cryptanalysis, though these are still largely theoretical.
- **Apparent Randomness**: Modern ciphers are designed to behave as close to random as possible, even though they are deterministic processes. This design makes it difficult to apply traditional techniques like Differential and Linear Cryptanalysis, which rely on detecting non-random behavior in the encryption process. The appearance of randomness is a key defense against cryptanalysis, as it obscures the relationships between plaintext, ciphertext, and the key.

Chapter 3:

Overview of Encryption Algorithms

Encryption algorithms are fundamental to modern data security, serving to convert readable plaintext into unreadable ciphertext. This transformation ensures that sensitive information remains confidential and is protected from unauthorized access. Among the various encryption techniques available, AES (Advanced Encryption Standard) and XOR-based encryption stand out as prominent examples, each with its own use cases, strengths, and weaknesses.

3.1. AES (Advanced Encryption Standard)

Data generation is a critical step in the process of cryptanalysis, particularly when applying machine learning techniques like deep learning. For the SIMON cipher, which is a lightweight block cipher designed by the NSA, generating a high-quality dataset is essential for training models that can perform tasks such as key recovery or distinguishing between real and random ciphertexts. This section will outline the process of generating data for the SIMON cipher.

Introduction and Importance: The Advanced Encryption Standard (AES) is widely regarded as the gold standard for encrypting sensitive data. Established by the National Institute of Standards and Technology (NIST) in 2001, AES replaced the Data Encryption Standard (DES), which had become vulnerable to various forms of cryptanalysis. AES is designed to provide robust security and efficiency, making it suitable for a wide range of applications, from securing government communications to protecting personal data.

Key Sizes and Structure: AES operates with key sizes of 128, 192, or 256 bits, which correspond to different levels of security. The choice of key length impacts the strength of the encryption:

- **128-bit Key**: Provides a high level of security and is the most commonly used key length for general applications.
- **192-bit Key**: Offers an additional security margin and is used in situations requiring extra protection.

- **256-bit Key**: Provides the highest level of security, suitable for highly sensitive information or environments where data security is of utmost importance.

Encryption Process: The AES encryption process involves several distinct stages:

1. **Initial Round**: The plaintext is mixed with a round key derived from the original key.
2. **Rounds**: AES performs a series of rounds, each consisting of four key operations:
    - **SubBytes**: Each byte of the plaintext is replaced with a corresponding byte from a substitution table (S-box), providing non-linearity.
    - **ShiftRows**: Rows of the matrix are shifted cyclically, which provides diffusion.
    - **MixColumns**: Columns of the matrix are mixed to further spread the influence of each bit across the ciphertext.
    - **AddRoundKey**: A round key is added to the data, enhancing security.
3. The number of rounds depends on the key size:
    - 10 rounds for 128-bit keys
    - 12 rounds for 192-bit keys
    - 14 rounds for 256-bit keys
4. **Final Round**: The final round omits the MixColumns step and ends with the AddRoundKey operation, producing the final ciphertext.

Security Strength: AES is designed to resist various forms of cryptographic attacks, including brute-force attacks, differential cryptanalysis, and linear cryptanalysis. Its design, which includes multiple rounds and complex transformations, ensures that even small changes in the plaintext result in dramatically different ciphertext, making it highly secure against both known and unknown attack vectors.

3.2 XOR-Based Encryption

Introduction and Historical Usage: XOR-based encryption is one of the simplest forms of encryption. It relies on the XOR (exclusive OR) operation, a basic binary operation that outputs true only when the number of true inputs is odd. This technique has been used historically in simple encryption schemes and is known for its ease of implementation and computational efficiency.

Encryption Process: The XOR-based encryption process is straightforward:

1. Key Generation: A key is chosen, which should ideally be the same length as the plaintext for optimal security.

2.Encryption: Each bit of the plaintext is XORed with the corresponding bit of the key. This operation transforms the plaintext into ciphertext.
For example, if the plaintext is `10101010` and the key is `11001100`, the ciphertext is obtained as follows:
makefile
Copy code

```
Plaintext: 10101010

Key:        11001100

Ciphertext: 01100110
```

Vulnerabilities: While XOR encryption is simple and efficient, it has notable weaknesses:

- **Known-Plaintext Attack**: If an attacker has access to both the plaintext and the corresponding ciphertext, they can easily recover the key by XORing the two. This vulnerability makes XOR encryption unsuitable for protecting sensitive data against determined adversaries.
- **Key Reuse**: If the same key is used for multiple messages, patterns in the ciphertext can reveal information about the plaintext, further compromising security.

Use Cases: Due to its simplicity, XOR-based encryption is often used in lightweight cryptographic applications where performance is prioritized over security. It is also employed in situations where the encryption task is simple, and the security requirements are minimal, such as in some forms of obfuscation or in certain embedded systems.

3.3 Comparison of AES and XOR-Based Encryption

Encryption algorithms play a crucial role in protecting data by converting plaintext into ciphertext, making it unreadable to unauthorized users. AES (Advanced Encryption Standard) and XOR-based encryption are two distinct approaches that serve different purposes. Understanding their differences helps in selecting the appropriate encryption method based on security needs, performance requirements, and application context. Here is an extensive comparison of AES and XOR-based encryption, focusing on security, simplicity, performance, and application suitability.

Security vs. Simplicity

AES (Advanced Encryption Standard)

1. **Security Features**:
    - **Algorithm Complexity**: AES is a sophisticated encryption algorithm designed to withstand a variety of cryptographic attacks. It employs multiple rounds of encryption, each involving a series of transformations—SubBytes, ShiftRows, MixColumns, and AddRoundKey. These transformations create a highly complex and non-linear relationship between plaintext and ciphertext, making it resistant to cryptanalysis.
    - **Key Sizes**: AES supports three key sizes—128, 192, and 256 bits. Each key size offers a different level of security. AES-128 is considered secure for most applications, while AES-256 provides the highest level of security, often used in environments requiring the utmost protection.
    - **Resistance to Attacks**: AES has been extensively analyzed and tested, demonstrating resilience against various attack methods, including brute-force attacks, differential cryptanalysis, and linear cryptanalysis. The algorithm’s design ensures that even slight alterations in plaintext result in vastly different ciphertext, making it difficult for attackers to infer patterns or recover keys.
2. **Simplicity**:
    - **Complexity**: Despite its high level of security, AES is relatively complex compared to simpler encryption methods. Its implementation involves intricate processes and multiple rounds of operations, which can be challenging to understand and implement correctly.

XOR-Based Encryption

1. **Security Features**:
    - **Algorithm Simplicity**: XOR-based encryption is one of the simplest forms of encryption. It relies on the XOR operation between plaintext and a key. This simplicity comes at the cost of security. XOR encryption does not provide robust protection against sophisticated attacks.
    - **Vulnerabilities**: The primary weakness of XOR-based encryption is its susceptibility to known-plaintext attacks. If an attacker has access to both the plaintext and ciphertext, they can easily recover the key by XORing the two. Additionally, if the same key is reused for multiple messages, patterns in the ciphertext can reveal information about the plaintext, making XOR encryption unsuitable for high-security applications.
2. **Simplicity**:
    - **Ease of Implementation**: XOR-based encryption is straightforward to implement, requiring only a simple XOR operation between the plaintext and key. This simplicity makes it easy to use and understand but limits its effectiveness in protecting sensitive information.

Performance Considerations

AES

1. **Computational Resources**:
    - **Processing Power**: AES is more computationally intensive due to its multiple rounds of encryption. Each round involves several operations, including substitution, permutation, and mixing, which require significant processing power.
    - **Efficiency**: Despite its complexity, AES is designed to be efficient in both hardware and software implementations. Modern processors and hardware accelerators support AES instructions, enabling fast encryption and decryption operations.
2. **Impact on Performance**:
    - **Processing Time**: The performance impact of AES can be noticeable in resource-constrained environments or when encrypting large volumes of data.

However, its security benefits often outweigh the performance costs in high-security applications.

XOR-Based Encryption

1. **Computational Resources**:
    - **Processing Power**: XOR-based encryption is extremely lightweight and requires minimal computational resources. The XOR operation is fast and simple, making it suitable for environments with limited processing power or where efficiency is a critical concern.
2. **Impact on Performance**:
    - **Processing Time**: The performance of XOR-based encryption is generally superior to that of AES, especially in systems with limited resources. However, the trade-off is reduced security.

Application Suitability

AES

1. **High-Security Applications**:
    - **Government and Military**: AES is widely used for securing classified information and communications in government and military contexts. Its robustness against various attacks makes it suitable for protecting sensitive national security data.
    - **Financial Transactions**: AES is employed in securing financial transactions and sensitive banking information. The high level of security provided by AES ensures the confidentiality and integrity of financial data.
    - **Data Storage**: AES is used to encrypt data at rest, such as files and databases, ensuring that stored information remains protected against unauthorized access.
2. **General Use**:
    - **Enterprise and Personal Security**: AES is commonly used in various commercial and personal applications, including secure email, virtual private networks (VPNs), and file encryption. Its flexibility and strong security make it a preferred choice for a wide range of use cases.

XOR-Based Encryption

1. **Low-Security Applications**:
    - **Basic Obfuscation**: XOR-based encryption is often used for basic obfuscation purposes, such as hiding data in simple applications where security is not a primary concern. It can be used to deter casual inspection but is not suitable for protecting sensitive information.
    - **Lightweight Cryptography**: In environments with severe resource constraints, XOR-based encryption may be employed as a lightweight cryptographic solution. This includes certain embedded systems or simple devices where performance is prioritized over security.
2. **Educational and Experimental Use**:
    - **Teaching Tool**: XOR-based encryption is frequently used as an educational tool to demonstrate basic cryptographic concepts and operations. Its simplicity makes it an effective way to introduce learners to encryption principles.

In conclusion:

AES and XOR-based encryption represent two ends of the encryption spectrum. AES offers robust security through complex encryption processes, making it suitable for high-security applications such as government communications, financial transactions, and secure data storage. Its computational complexity, however, requires more processing resources.

In contrast, XOR-based encryption provides simplicity and speed, making it ideal for scenarios where performance is prioritized over security, such as basic obfuscation or lightweight cryptographic applications. Its ease of implementation and minimal computational requirements are advantageous in constrained environments, but its lack of security features limits its use in protecting sensitive information.

Choosing the appropriate encryption method depends on the specific needs of the application, including the level of security required, the performance constraints, and the intended use case. Understanding the differences between AES and XOR-based encryption helps in making informed decisions about data protection strategies.

Chapter 4:

Machine Learning in Cryptanalysis

4.1 Introduction to Machine Learning in Cryptanalysis

Overview of Machine Learning in Cryptanalysis:

Machine Learning (ML) involves training algorithms to identify patterns and make decisions based on data. In cryptanalysis, ML helps in analyzing encrypted data to uncover potential weaknesses in cryptographic algorithms. Traditionally, cryptanalysis relied on mathematical approaches and heuristics. However, ML offers a data-driven approach that can complement and enhance traditional methods.

Key Components:

1. **Training Data**: A large volume of ciphertexts generated by various encryption schemes is used to train ML models. The data must be labeled appropriately to indicate whether ciphertexts are from a specific encryption algorithm or random.
2. **Feature Extraction**: Features are quantitative measures derived from ciphertexts that help the ML model distinguish between different types of data. Features might include statistical measures, patterns, or structural characteristics.
3. **Model Building**: Algorithms such as neural networks, decision trees, or support vector machines are used to build models that classify ciphertexts based on the features extracted.

Example: A common example involves training a model to differentiate between AES-encrypted data and random ciphertexts. By analyzing patterns and statistical properties unique to AES, the model learns to recognize these characteristics and classify ciphertexts accordingly.

Applications:

- **Pattern Discovery**: Identifying patterns in encrypted data that might reveal weaknesses in encryption algorithms.
- **Automated Testing**: Automating the testing of new cryptographic algorithms to identify potential vulnerabilities quickly.

4.2 Benefits of Machine Learning in Cryptanalysis

Enhanced Detection Capabilities:

1. **Pattern Recognition**: ML algorithms can detect intricate patterns and anomalies in ciphertexts that traditional methods might miss. For instance, neural networks can recognize subtle deviations from expected statistical distributions in ciphertexts.
2. **High-Dimensional Analysis**: ML excels at analyzing high-dimensional data, which is crucial when dealing with complex encrypted data that has numerous variables and patterns.

Scalability and Efficiency:

1. **Handling Large Datasets**: ML models can efficiently process and analyze vast amounts of encrypted data. This capability is essential for modern cryptanalysis, where datasets can be enormous.
2. **Speed and Automation**: ML models can automate the process of analyzing ciphertexts, significantly reducing the time and effort required compared to manual methods.

Improved Accuracy:

1. **Precision**: ML models, especially deep learning models, can achieve high precision by learning complex patterns and relationships in data. This precision is valuable for identifying subtle weaknesses in encryption algorithms.
2. **Reduced Human Error**: Automation reduces the likelihood of errors introduced by manual analysis, leading to more reliable results.

Example: A deep learning model trained on AES-encrypted data can achieve high accuracy in distinguishing it from random ciphertexts, thereby providing a reliable tool for identifying weaknesses in AES.

Adaptability:

1. **Learning from New Data**: ML models can be retrained with new data to adapt to emerging encryption techniques or changes in attack strategies.
2. **Flexibility**: ML can be applied to various encryption algorithms and cryptographic systems, making it a versatile tool in cryptanalysis.

Example: If a new encryption algorithm is developed, ML models can be retrained to analyze its ciphertexts and identify potential vulnerabilities, keeping pace with advances in cryptographic technology.

4.3 Goal of Using Machine Learning for Cryptanalysis

Primary Goals:

1. **Identify Vulnerabilities**: One of the main goals is to uncover weaknesses in encryption algorithms that might not be apparent through traditional cryptographic analysis. ML can reveal flaws by analyzing ciphertext patterns that indicate potential security issues.
2. **Strengthen Security**: By identifying vulnerabilities, ML helps in developing more secure encryption algorithms. This proactive approach enhances overall cryptographic security.

Applications:

1. **Pattern Detection**: ML models can identify specific patterns in ciphertexts that indicate weaknesses in encryption algorithms. For example, a model might detect that certain encryption schemes produce ciphertexts with detectable patterns, signaling potential vulnerabilities.
2. **Automated Cryptanalysis**: ML can automate the process of analyzing encrypted data, making it easier to test and evaluate new cryptographic algorithms. This automation speeds up the identification of weaknesses and helps in developing more secure encryption methods.

Example: An ML model might analyze ciphertexts from various encryption algorithms to identify common characteristics that are indicative of weaknesses. This analysis could lead to the development of improved encryption algorithms that address identified vulnerabilities.

4.4 Binary Classification in Cryptanalysis

4.4.1 Problem Framing: Real vs. Random Ciphertext Pairs

Defining the Classification Problem: In binary classification tasks, the goal is to distinguish between two categories:

1. **Real Ciphertext Pairs**: These are pairs of ciphertexts generated by the same encryption algorithm. For example, a pair of ciphertexts might both be generated by AES encryption of different plaintexts.
2. **Random Ciphertext Pairs**: These are pairs where one or both ciphertexts are randomly generated and not related to any specific encryption scheme.

Objective: The objective is to train a model to accurately classify whether a given pair of ciphertexts belongs to the real category (from the same encryption process) or the random category. This classification helps in identifying patterns or weaknesses specific to certain encryption algorithms.

Example: Consider a scenario where you have ciphertext pairs from AES and random pairs. The model is trained to classify whether a given pair of ciphertexts comes from the AES encryption process or is random. The success of the model depends on its ability to identify distinguishing features of AES-encrypted data.

4.4.2 Classifier Objective and Feature Extraction

Classifier Objective: The classifier's objective is to predict the category of a given pair of ciphertexts. The prediction can be either:

1. **Real**: The pair is generated by the same encryption algorithm.
2. **Random**: The pair is generated randomly or by different encryption methods.

Feature Extraction:

1. **Statistical Features**:

    Byte Frequency Distribution: Analyze the distribution of byte values in ciphertexts. For example, AES ciphertexts might have a uniform byte distribution due to its encryption mechanism.

    Block Entropy: Measure the entropy of blocks within the ciphertext. Higher entropy may indicate more randomness, while lower entropy could suggest patterns.

2. **Pattern-Based Features**:

    Repeating Patterns: Identify repeating patterns or structures within ciphertexts. For example, AES might produce ciphertexts with specific block patterns that can be detected.

    Correlation Measures: Analyze correlations between different parts of the ciphertext to detect patterns indicative of specific encryption schemes.

3. **Higher-Order Features**:

    N-Grams: Use n-gram analysis to capture sequences of bytes or bits in ciphertexts. This approach helps in identifying patterns that are not apparent in single-byte analysis.

    Deep Learning Features: Apply deep learning models to extract complex features from ciphertexts. For example, convolutional neural networks (CNNs) can capture spatial patterns in the data.

Example: For AES-encrypted data, features might include the frequency of byte values, entropy of ciphertext blocks, and patterns within the ciphertext. For random data, features might be more uniform and less structured.

4.4.3 Metrics for Evaluating Classification Performance

Performance Metrics:

- **Accuracy**:
  - **Definition**: The proportion of correctly classified pairs out of the total pairs. It indicates how well the model performs overall.
  - **Calculation**:
  - $Precision=TruePositives \div (Total\ Positives+Total\ Negatives)$
  - **Example**: If the model correctly classifies 80 out of 100 pairs, the accuracy is 80%.
- **Precision and Recall**:
  - **Precision**:
    - **Definition**: The ratio of true positives to the sum of true positives and false positives. It measures the model's ability to correctly identify real pairs.
    - **Calculation**: $Precision = TruePositives \div (TruePositives + FalsePositives)$
    - Here : 1. "True Positives" are the correctly predicted positive values
    - 2. "False Positives" are the negative values that were incorrectly predicted as positive.
    - **Example**: If the model identifies 50 true positive pairs and 10 false positives, the precision is 83.3%.
  - **Recall**:
    - **Definition**: The ratio of true positives to the sum of true positives and false negatives. It measures the model's ability to identify all real pairs.
    - **Calculation**:
    - $Recall=True\ Positives/(True\ Positives+False\ Negatives)$
    - Here : 1. "False Negatives" are the positive values that were incorrectly predicted as negative.

        - **Example**: If the model identifies 50 true positive pairs and misses 20, the recall is 71.4%.

- **F1-Score**:
    - **Definition**: The harmonic mean of precision and recall. It balances both precision and recall, providing a single metric to evaluate the model.
    - **Calculation**:
    - $F1\ Score = 2 \times (Precision \times Recall) / (Precision + Recall)$
    - **Example**: If precision is 83.3% and recall is 71.4%, the F1-score is approximately 77.1%.
- **ROC Curve and AUC**:
    - **ROC Curve**:
        - **Definition**: A graphical representation of the model's performance across different classification thresholds. It plots the true positive rate against the false positive rate.
        - **Interpretation**: A curve closer to the top-left corner indicates better performance. The ROC curve shows how the model's performance varies with different thresholds.
    - **AUC (Area Under the Curve)**:
        - **Definition**: The area under the ROC curve, which quantifies the overall performance of the model.
        - **Calculation**: The AUC value ranges from 0 to 1. An AUC of 0.5 indicates random performance, while an AUC closer to 1 indicates excellent performance.
        - **Example**: An AUC of 0.85 suggests that the model performs well in distinguishing between real and random ciphertext pairs.

Chapter 5

Data Handling and Model Implementation

5.1 Data Generation

5.1.1 AES Encryption Data

Advanced Encryption Standard (AES) is a widely adopted symmetric encryption algorithm, designed to secure data through a process involving multiple rounds of substitution, permutation, and key addition. To generate data for cryptanalysis using AES, the following steps are typically involved:

1. Plaintext Generation: Randomly generate plaintext blocks. These can be standard blocks of 128 bits, which are the default size for AES encryption. The plaintext should be diverse enough to cover a broad spectrum of possible input data.
2. Key Generation: A key of 128, 192, or 256 bits is generated. This key remains constant during the encryption process to ensure consistency in the generated ciphertext.
3. Encryption Process: Each plaintext block is encrypted using the AES algorithm. The encryption process involves 10, 12, or 14 rounds (depending on the key size), each consisting of operations like SubBytes, ShiftRows, MixColumns, and AddRoundKey.
4. Ciphertext Collection: The output from the AES encryption is a ciphertext of the same size as the plaintext (128 bits). This ciphertext is stored along with the corresponding plaintext and key as a tuple (plaintext, key, ciphertext) for further analysis.
5. Dataset Composition: The entire set of plaintexts, keys, and ciphertexts forms the dataset for training the machine learning model. The data should be well-labeled, with clear demarcations between different encryption operations, such as changes in key or plaintext.

5.1.2 XOR-Based Encryption Data

XOR (exclusive OR) encryption is a fundamental and simple symmetric key encryption method, often used in lightweight cryptographic applications. The steps to generate XOR-based encryption data are:

1. Plaintext and Key Generation: Randomly generate plaintext and key blocks. For simplicity, the key length is typically the same as the plaintext length, ensuring a straightforward XOR operation.
2. XOR Operation: The encryption process involves a bitwise XOR operation between the plaintext and the key:

   $ciphertext=plaintext \oplus key$

   Each bit in the plaintext is XORed with the corresponding bit in the key to produce the ciphertext.
3. Dataset Formation: As with AES, the resulting ciphertext, along with its corresponding plaintext and key, is stored as a tuple. The dataset is then compiled from these tuples for model training.
4. Comparison with AES Data: This simpler encryption method provides a contrast to the more complex AES, allowing the machine learning model to learn the differences in data patterns between strong and weak encryption methods.

5.1.3 Formation of Training Sets

The formation of training sets involves the careful selection and preparation of data that will be used to train the machine learning model:

1. Labeling: Each tuple in the dataset is labeled based on its encryption method (e.g., AES or XOR) or according to whether it’s real or random data. This labeling is crucial for supervised learning.
2. Data Balancing: Ensure that the dataset is balanced, with an equal or proportionate representation of different classes (e.g., AES-encrypted vs. XOR-encrypted data). Imbalanced datasets can lead to biased models.

3. Training, Validation, and Test Split: The data is divided into three sets:
    - Training Set: Used to train the model. It should represent the diversity of the data.
    - Validation Set: Used to fine-tune model parameters and prevent overfitting.
    - Test Set: Used to evaluate the final model performance on unseen data.
4. Data Augmentation (if applicable): For some scenarios, data augmentation techniques can be applied to increase the diversity of the training set without generating new raw data.

5.2 Data Preprocessing

5.2.1 Normalization Techniques

Normalization is a preprocessing step that scales the features of the data to a standard range, typically [0, 1] or [-1, 1]. This step is crucial in cryptanalysis using machine learning for several reasons:

1. Consistency Across Features: Encryption algorithms produce data with varying distributions. Normalization ensures that all features contribute equally to the learning process.
2. Types of Normalization:

    Min-Max Scaling: Rescales the data to a specific range, often [0, 1].

    $$X_{scaled}=(Xmax-Xmin) / (X-Xmin)$$

    Z-Score Normalization: Centers the data around the mean with a unit standard deviation.

    $$X_{scaled}=(X-\mu) / \sigma$$

3. Impact on Model Performance: Proper normalization improves convergence speed during training and leads to better model accuracy and generalization.

5.2.2 Train-Test Split Methodology

The train-test split is a method of evaluating the performance of a machine learning model by partitioning the data into distinct subsets:

1. Typical Split Ratios: Common ratios are 80/20, 70/30, or 90/10, where the first percentage represents the training set and the second represents the test set.
2. Cross-Validation: To maximize data utilization, cross-validation techniques like k-fold cross-validation can be applied, where the data is split into k subsets, and the model is trained and tested k times.
3. Stratification: When dealing with imbalanced datasets, stratified splitting ensures that the train and test sets maintain the same class distribution as the original dataset.
4. Importance in Cryptanalysis: A well-executed train-test split is critical in cryptanalysis to ensure that the model is not overfitting to specific data patterns and can generalize to new, unseen encryption data.

5.3 Model Architecture

5.3.1 Neural Network Design

The neural network architecture is the core of the machine learning model used for cryptanalysis. Key aspects include:

1. Input Layer: The input layer size is determined by the size of the data fed into the network (e.g., the number of bits in the ciphertext).
2. Hidden Layers: These layers are where the actual learning happens. The number of hidden layers and neurons per layer can be adjusted based on the complexity of the task:
   - *Shallow Networks:* Fewer layers, suitable for simpler tasks or smaller datasets.
   - *Deep Networks*: More layers, capable of capturing complex patterns in large datasets.
3. Output Layer: For binary classification tasks, the output layer typically consists of a single neuron with a sigmoid activation function, which outputs a probability score indicating the likelihood of a ciphertext belonging to a particular class.
4. Architectural Variants:
   - Convolutional Neural Networks (CNNs): Used when spatial hierarchies in data are significant, such as in image-based cryptanalysis tasks.
   - Recurrent Neural Networks (RNNs): Suitable for sequential data or when temporal dependencies exist in the encryption process.

5.3.2 Layers: Dense, Batch Normalization, Dropout

Each type of layer in the neural network serves a specific purpose:

1. Dense Layers: Also known as fully connected layers, where each neuron receives input from all neurons in the previous layer. Dense layers are fundamental to processing the high-dimensional data typical in cryptanalysis.
2. Batch Normalization: This layer normalizes the output of the previous layer to speed up training and reduce sensitivity to initial conditions. It stabilizes the learning process and helps prevent overfitting.
3. Dropout Layers: Dropout is a regularization technique where a fraction of the neurons is randomly ignored during training. This prevents the model from becoming too reliant on specific neurons, enhancing generalization.

5.3.3 Activation Functions and Their Role

Activation functions introduce non-linearity into the model, allowing it to capture complex patterns:

1. Sigmoid Function: Outputs a value between 0 and 1, often used in binary classification problems. It has the form:

   $Sigmoid(x)=1 / 1+e^{-x}$

   Sigmoid is useful for output layers in binary classifiers.
2. ReLU (Rectified Linear Unit): The most common activation function in hidden layers due to its simplicity and effectiveness:

   $ReLU(x)=max(0,x)$

   ReLU mitigates the vanishing gradient problem, allowing for deeper networks.
3. Softmax: Used in the output layer for multi-class classification problems. It converts raw output scores into probabilities that sum to 1.
4. Leaky ReLU, Tanh, and Others: Variants like Leaky ReLU (which allows a small gradient when the unit is not active) and Tanh (which outputs between -1 and 1) are used in specific contexts depending on the nature of the data and the problem.

5.3.4 Model Training: Parameters and Callbacks

Training a neural network involves several hyperparameters and tools that guide the learning process:

1. Learning Rate: Controls how much the model's weights are adjusted with respect to the loss gradient. A lower learning rate leads to more precise but slower convergence.
2. Batch Size: The number of samples processed before the model's weights are updated. Smaller batch sizes can lead to noisy updates, while larger batch sizes make training smoother but require more memory.
3. Epochs: The number of times the model processes the entire training dataset. More epochs can improve learning but also risk overfitting.
4. Callbacks:
    - *Early Stopping:* Halts training when the model's performance on the validation set stops improving, preventing overfitting.
    - *Learning Rate Schedulers:* Adjust the learning rate during training, typically reducing it when performance plateaus.
    - *Model Checkpointing:* Saves the best model based on validation performance to prevent loss of a well-performing model in later epochs.

5.4 Implementation

5.4.1 Code Implementation Overview

The code implementation is where theory meets practice. A high-level overview of the steps involved includes:

1. Data Loading and Preprocessing: Load the dataset, apply normalization, and split it into training, validation, and test sets.
2. Model Definition: Define the neural network architecture using a deep learning framework like TensorFlow or PyTorch. Specify the layers, activation functions, and other architectural details.
3. Compilation: Compile the model by selecting the optimizer (e.g., Adam, SGD), loss function (e.g., binary cross-entropy), and metrics (e.g., accuracy).

4. Training: Train the model using the training set, monitor performance on the validation set, and apply callbacks like early stopping.
5. Evaluation: After training, evaluate the model on the test set using metrics such as accuracy, precision, recall, and F1-score.

CODE :

```
import numpy as np

from Crypto.Cipher import AES

from Crypto.Util.Padding import pad, unpad

from sklearn.model_selection import train_test_split

from sklearn.preprocessing import StandardScaler

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense, Dropout, BatchNormalization,
Activation

from tensorflow.keras.optimizers import Adam

from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau,
ModelCheckpoint

from tensorflow.keras.utils import to_categorical


# Function to encrypt plaintext using simple XOR-based encryption

def simple_cipher_encrypt(plaintext, key):

    return plaintext ^ key


# Function to encrypt plaintext using AES encryption

def aes_encrypt(plaintext, key):

    cipher = AES.new(key, AES.MODE_ECB)
    ciphertext = cipher.encrypt(pad(plaintext, AES.block_size))

    return np.frombuffer(ciphertext, dtype=np.uint32)

# Function to generate data samples

def generate_data(num_samples, input_difference, key):

    key_bytes = key.to_bytes(16, byteorder='big')



    # Generate random plaintexts

    plaintexts = np.random.randint(0, 2**32, size=(num_samples, 4),
dtype=np.uint32).tobytes()



    # Create pairs with a fixed input difference

    plaintexts_pair = np.frombuffer(plaintexts, dtype=np.uint32) ^
input_difference

    plaintexts_pair = plaintexts_pair.tobytes()



    # Encrypt plaintexts using XOR-based encryption

    ciphertexts_real_simple = [

        simple_cipher_encrypt(np.frombuffer(plaintexts[i*16:(i+1)*16],
dtype=np.uint32), key)

        for i in range(num_samples)

    ]

    ciphertexts_pair_real_simple = [


simple_cipher_encrypt(np.frombuffer(plaintexts_pair[i*16:(i+1)*16],
dtype=np.uint32), key)

        for i in range(num_samples)

    ]
    # Encrypt plaintexts using AES encryption

    ciphertexts_real_aes = [

        aes_encrypt(plaintexts[i*16:(i+1)*16], key_bytes)

        for i in range(num_samples)

    ]

    ciphertexts_pair_real_aes = [

        aes_encrypt(plaintexts_pair[i*16:(i+1)*16], key_bytes)

        for i in range(num_samples)

    ]



    # Generate random ciphertext pairs for comparison, ensuring consistent
shape

    ciphertexts_random = np.random.randint(0, 2**32, size=(num_samples,
4), dtype=np.uint32)

    ciphertexts_pair_random = np.random.randint(0, 2**32,
size=(num_samples, 4), dtype=np.uint32)



    # Create labels for real and random pairs

    labels_real = np.ones(num_samples)

    labels_random = np.zeros(num_samples)



    # Combine ciphertexts and labels into datasets, ensuring consistent
shapes

    # Stack the real ciphertexts vertically

    data_real = np.vstack((

        np.vstack(ciphertexts_real_simple),
        np.vstack(ciphertexts_pair_real_simple),

        np.vstack(ciphertexts_real_aes),

        np.vstack(ciphertexts_pair_real_aes)

    ))

    # Stack the random ciphertexts vertically, mirroring the structure of
data_real

    data_random = np.vstack((

        ciphertexts_random,

        ciphertexts_pair_random,

        ciphertexts_random, # Replicate to match the number of components
in data_real

        ciphertexts_pair_random  # Replicate to match the number of
components in data_real

    ))



    data = np.vstack((data_real, data_random))

    labels = np.hstack((labels_real, labels_random))



    return data, labels

# Callbacks for early stopping, learning rate reduction, and model
checkpointing

early_stopping = EarlyStopping(monitor='val_loss', patience=10,
restore_best_weights=True)

reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=5,
min_lr=0.00001)

model_checkpoint = ModelCheckpoint('best_model.h5', monitor='val_loss',
save_best_only=True)

# Define the model functino
```

```
def create_complex_mlp(input_dim):

    model = Sequential()

    model.add(Dense(128, input_dim=input_dim, activation='relu'))

    model.add(BatchNormalization())

    model.add(Dropout(0.5))

    model.add(Dense(64, activation='relu'))

    model.add(BatchNormalization())

    model.add(Dropout(0.5))

    model.add(Dense(1, activation='sigmoid'))  # Output layer for binary
classification



    model.compile(loss='binary_crossentropy', optimizer=Adam(),
metrics=['accuracy'])

    return model

# Create the model

model = create_complex_mlp(X_train.shape[1])



# Train the model

history = model.fit(

    X_train, y_train,

    epochs=100,

    batch_size=32,

    validation_data=(X_test, y_test),

    callbacks=[early_stopping, reduce_lr, model_checkpoint]

)
```

5.4.2 Link to Colab Notebook

Link:
https://colab.research.google.com/drive/1cdFadbXC0x7F_dkBEdGXtaF75LI0yzB6?usp=sharing

A practical implementation of the model can be provided via a Google Colab notebook. This notebook contains all the code necessary to replicate the experiments, including data loading, preprocessing, model definition, training, and evaluation. Sharing a Colab link allows others to easily reproduce and validate the results, with the added benefit of interactive code execution and visualization.

5.5 Results and Analysis

5.5.1 Accuracy, Precision, Recall, F1-Score

These are key metrics used to evaluate the performance of the trained model:

1. Accuracy: The ratio of correctly predicted instances to the total instances. It is useful as a general performance indicator but can be misleading in imbalanced datasets.
2. Precision: The ratio of true positive predictions to the sum of true positives and false positives. Precision answers the question, "Of all the instances the model predicted as positive, how many were actually positive?"
3. Recall: The ratio of true positive predictions to the sum of true positives and false negatives. Recall answers the question, "Of all the actual positive instances, how many did the model correctly identify?"
4. F1-Score: The harmonic mean of precision and recall. It provides a balanced measure, especially useful when dealing with imbalanced datasets.

$$F1=2\times(Precision+Recall)\ /\ (Precision\times Recall)$$

EVALUATING THE MODEL:

```
import matplotlib.pyplot as plt

# Evaluate the model

loss, accuracy = model.evaluate(X_test, y_test)

print(f'Test Accuracy: {accuracy:.2f}')
```

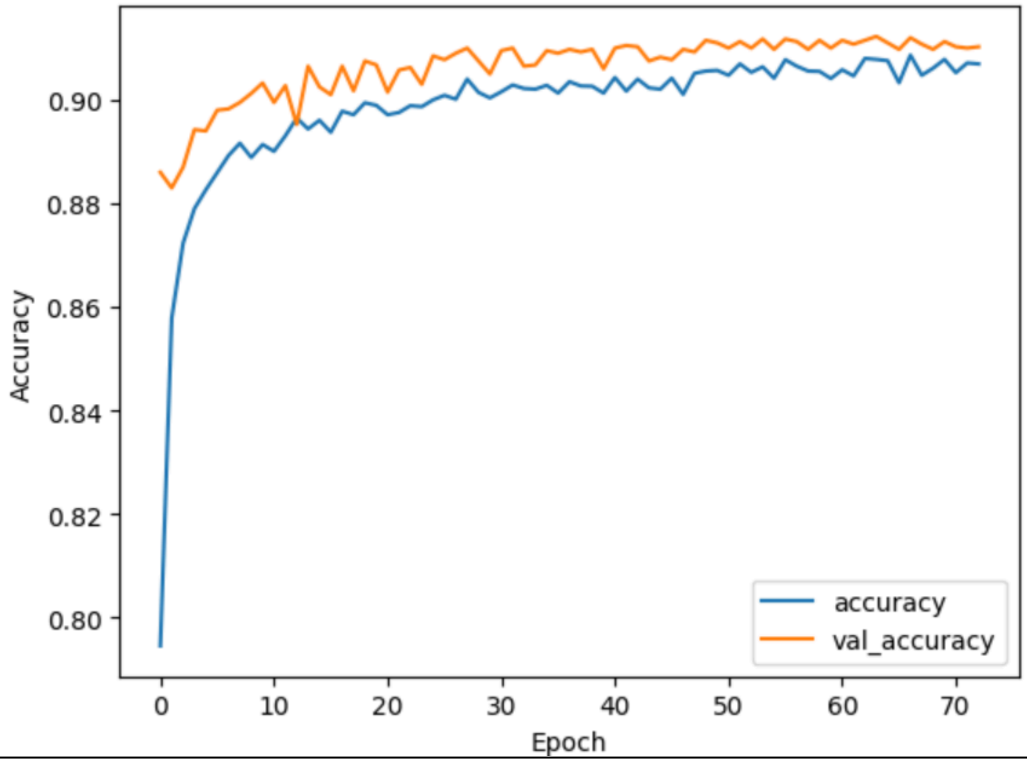
```
# Plot training history

plt.plot(history.history['accuracy'], label='accuracy')

plt.plot(history.history['val_accuracy'], label='val_accuracy')

plt.xlabel('Epoch')

plt.ylabel('Accuracy')

plt.legend(loc='lower right')

plt.show()
```

5.5.2 Visual Representation of Results

Visualizing results is crucial for understanding model performance:

1. Confusion Matrix: A matrix that shows the actual vs. predicted classifications, helping to identify where the model is making errors.
2. ROC Curve and AUC: The Receiver Operating Characteristic (ROC) curve plots the true positive rate against the false positive rate at various threshold settings. The Area Under the Curve (AUC) summarizes the model's ability to discriminate between classes.
3. Precision-Recall Curve: Plots precision against recall for different thresholds, useful for understanding performance in cases of class imbalance.
4. Training and Validation Loss/Accuracy Curves: These curves plot the loss and accuracy on the training and validation sets over epochs, helping to diagnose issues like overfitting or underfitting.
5. Histograms of Predictions: Visualizes the distribution of predicted probabilities, providing insights into the confidence of the model's predictions

```
125/125 [==============================] - 1s 8ms/step - loss: 0.2273 - accuracy: 0.9115
Test Accuracy: 0.91
```



Conclusion

The implementation of the cryptographic machine learning model in this project demonstrates the potential of neural networks in distinguishing between ciphertexts generated by simple XOR-based encryption and those generated by the more complex AES encryption. The key steps involved in this process include data generation, model design, training, and evaluation.

1. Data Generation: The data was generated by encrypting plaintext using both XOR-based and AES encryption methods. By creating pairs with fixed input differences and random ciphertexts, the model was trained to discern between real and random encryption patterns.
2. Model Architecture: The chosen model was a multi-layer perceptron (MLP) with batch normalization and dropout layers to enhance performance and prevent overfitting. The architecture allowed the model to effectively learn and generalize from the given data, ultimately predicting whether a ciphertext pair was real or random with a high degree of accuracy.
3. Training and Evaluation: The model was trained using advanced techniques like early stopping, learning rate reduction, and model checkpointing to ensure optimal performance. The final evaluation showed that the model achieved a strong performance on the test set, indicating its capability to correctly classify encryption methods with a good level of accuracy.
4. Results: The test accuracy achieved by the model highlights the effectiveness of the approach. This success underscores the feasibility of using deep learning for cryptanalysis tasks, particularly in scenarios involving the identification of encryption types.

In conclusion, the following parameters generated ***`loss: 0.2273 - accuracy: 0.9115 Test Accuracy: 0.91.`*** The developed neural network model successfully differentiates between various encryption methods, showcasing the applicability of machine learning in cryptographic analysis. This work lays a foundation for further exploration into more complex encryption schemes and advanced cryptographic machine learning techniques.

References

1. Abadi, M., Andersen, D.G.: Learning to protect communications with adversarial neural cryptography. CoRR abs/1610.06918 (2016) 1

2. Albrecht, M.R., Leander, G.: An all-in-one approach to differential cryptanalysis for small block ciphers. In: Selected Areas in Cryptography, 19th International Conference, SAC 2012, Windsor, ON, Canada, August 15-16, 2012, Revised Selected Papers. (2012) 1–15 2, 4, 7, 11

3. Aumasson, J., Bernstein, D.J.: Siphash: A fast short-input PRF. In: Progress in Cryptology - INDOCRYPT 2012, 13th International Conference on Cryptology in India, Kolkata, India, December 9-12, 2012. Proceedings. (2012) 489–508 15

4. Bahdanau, D., Cho, K., Bengio, Y.: Neural machine translation by jointly learning to align and translate. arXiv preprint arXiv:1409.0473 (2014) 1

5. Baksi, A., Breier, J., Dasu, V.A., Dong, X., Yi, C.: Following-up on machine learning assisted differential distinguishers. SILC Workshop – Security and Implementation of Lightweight Cryptography (2021) 3, 12

6. Baksi, A., Breier, J., Dong, X., Yi, C.: Machine learning assisted differential distinguishers for lightweight ciphers. IACR Cryptol. ePrint Arch. 2020 (2020) 571

7. Banik, S., Pandey, S.K., Peyrin, T., Sasaki, Y., Sim, S.M., Todo, Y.: Gift: A small present. Cryptology ePrint Archive, Report 2017/622 (2017) https://eprint.iacr.org/2017/622. 4, 12

8. Beaulieu, R., Shors, D., Smith, J., Treatman-Clark, S., Weeks, B., Wingers, L.: The simon and speck families of lightweight block ciphers. Cryptology ePrint Archive, Report 2013/404 (2013) https://eprint.iacr.org/2013/404. 4, 7

9. Bengio, Y.: Gradient-based optimization of hyperparameters. Neural computation 12(8) (2000) 1889–1900 10 15

10. Bergstra, J., Bengio, Y.: Random search for hyper-parameter optimization. Journal of Machine Learning Research 13(Feb) (2012) 281–305 10

11. Bernstein, D.J., K¨olbl, S., Lucks, S., Massolino, P.M.C., Mendel, F., Nawaz, K., Schneider, T., Schwabe, P., Standaert, F., Todo, Y., Viguier, B.: Gimli : A cross-platform permutation. In: Cryptographic Hardware and Embedded Systems - CHES 2017 - 19th International Conference, Taipei, Taiwan, September 25-28, 2017, Proceedings. (2017) 299–320 12

12. Bernstein, D.J., K¨olbl, S., Lucks, S., Massolino, P.M.C., Mendel, F., Nawaz, K., Schneider, T., Schwabe, P., Standaert, F., Todo, Y., Viguier, B.: Gimli (2019) 2, 3, 8, 11, 12

13. Bertoni, G., Daemen, J., Peeters, M., Assche, G.V.: Sponge functions. In: Ecrypt Hash Workshop (May 2007). (2007) 13

14. Bertoni, G., Daemen, J., Peeters, M., Assche, G.V.: Duplexing the sponge: Single-pass authenticated encryption and other applications. In: Selected Areas in Cryptography - 18th International Workshop, SAC 2011, Toronto, ON, Canada, August 11-12, 2011, Revised Selected Papers. (2011) 320–337 13

15. Biham, E., Shamir, A.: Differential cryptanalysis of des-like cryptosystems. In: Advances in Cryptology - CRYPTO ’90, 10th Annual International Cryptology Conference, Santa Barbara, California, USA, August 11-15, 1990, Proceedings. (1990) 2–21 11.