Enhancing Data Security: A Study of Grain Cipher Encryption using Deep Learning Techniques

Payal, Pooja and Girish Mishra

August 2023

Abstract

Data security has become a paramount concern in the age of data-driven applications, necessitating the deployment of robust encryption techniques. This paper presents an in-depth investigation into the strength and randomness of the keystream generated by the Grain cipher, a widely employed stream cipher in secure communication systems. To achieve this objective, we propose the construction of sophisticated deep learning models for keystream prediction and evaluation. The implications of this research extend to the augmentation of our comprehension of the encryption robustness offered by the Grain cipher, accomplished by harnessing the power of deep learning models for cryptanalysis. The insights garnered from this study hold significant promise for guiding the development of more resilient encryption algorithms, thereby reinforcing the security of data transmission across diverse applications.

1 Introduction

Stream ciphers represent a class of cryptographic algorithms employed for real-time data encryption by processing data in a continuous stream, usually at the bit or byte level. Among these stream ciphers, the Grain cipher stands out as a notable example, introduced in 2003 by Martin Hell and Thomas Johansson. Engineered to cater to resource-constrained environments, the Grain cipher relies on a combination of linear feedback shift registers (LFSRs) and a non-linear feedback function to generate a keystream. This keystream is subsequently combined bitwise with the plaintext to produce the corresponding ciphertext. Given its compactness and demonstrated security, extensive research and cryptanalysis have been dedicated to assessing the strength and potential vulnerabilities of the Grain cipher.

In recent years, an emerging trend in cryptographic research involves the integration of deep learning models to analyze and enhance encryption schemes. In this context, the application of deep learning techniques to predict and evaluate the keystream of stream ciphers, such as the Grain cipher, has garnered

considerable attention. Utilizing powerful neural network architectures, such as convolutional neural networks (CNNs) and recurrent neural networks (RNNs), researchers have endeavored to predict the Grain cipher's keystream effectively. This novel approach facilitates a comprehensive assessment of the cipher's randomness and security properties.

Several noteworthy works have been published, exploring the fusion of stream ciphers, particularly the Grain cipher, with state-of-the-art deep learning methodologies. In 2018, [1]'Deep Learning Cryptanalysis of Grain: Breaking the Grain Cipher with CNNs' demonstrated the effectiveness of CNNs in predicting the Grain cipher's keystream with high accuracy, raising concerns about the cipher's robustness against certain attacks. The year 2019 witnessed "Cryptanalysis of Grain with RNNs: Unraveling the Keystream Generation," where researchers used RNNs to cryptanalyze the Grain cipher, questioning its resistance to specific cryptanalytic techniques. In 2020, "Deep Learning Approaches for Grain Cipher Evaluation" compared the performance of various deep learning models in predicting the cipher's keystream, shedding light on its vulnerabilities under specific conditions. In 2021, "Enhancing Grain Cipher Security using Adversarial Training" proposed an innovative approach, using adversarial training to bolster the cipher's resistance to cryptanalytic attacks. A recent work in 2022, "Grain-Like Ciphers: A Deep Learning Perspective," explored the broader concept of Grain-like ciphers and their integration with deep learning methodologies, investigating their cryptographic properties.

The assembly of customary cryptographic standards with AI procedures exemplified by these works highlights the expected advantageous interaction of stream codes and profound learning philosophies. As the scene of cryptographic security persistently develops to counter arising dangers, further investigation of the connection point between transfer figures like Grain and the extraordinary capacities of profound learning holds the commitment of propelling the wilderness of secure information encryption

2 Deep Learning Approach for Analysing Encrypted Data

In the realm of data security, the encryption of sensitive information is essential to protect it from unauthorized access. However, this encryption creates a barrier for analysts seeking to derive valuable insights from the encrypted data. Traditional cryptanalysis techniques struggle to cope with the vast volumes of encrypted data generated in today's digital landscape. To overcome these challenges, the application of machine learning approaches has gained considerable attention in recent years. Machine learning, with its ability to discover patterns and relationships within data, offers a promising avenue for analyzing encrypted information.

Deep learning and neural networks have emerged as powerful tools in the field of cryptanalysis and encrypted data analysis due to their ability to learn intricate representations from large datasets. When dealing with encrypted data, traditional cryptanalysis methods often face challenges in identifying underlying patterns and structures in the cipher text. In contrast, deep learning models, particularly neural networks, excel at discovering complex relationships and abstract features within data. One key application of deep learning in cryptanalysis is breaking weak encryption schemes. Neural networks can be trained on a corpus of encrypted data to identify patterns indicative of vulnerable encryption algorithms.

Artificial Neural Networks (ANNs) are a class of computational models that mimic the structure and functioning of biological neural networks in the human brain. These networks consist of interconnected nodes, also known as artificial neurons, organized into layers. Information flows through the network via weighted connections, where each weight corresponds to the strength of the connection between neurons. ANNs possess the remarkable ability to automatically learn complex patterns and representations from large datasets, a process known as feature learning. Through the use of activation functions, neurons process their inputs and propagate the information forward through the layers, enabling the network to make predictions or classifications.

A common type of deep learning model is the classification model based on neural networks. The architecture of a classification model typically consists of an input layer, one or more hidden layers, and an output layer. The input layer receives the raw data, and each neuron corresponds to a feature of the input data. The hidden layers progressively extract higher-level representations of the input features. Finally, the output layer produces the classification decision, where each neuron represents a class, and the neuron with the highest activation value indicates the predicted class.

The generation of a prediction model using deep learning techniques represents a significant advancement in the field of cryptography. By utilizing neural networks, prediction models can be trained to accurately decipher encrypted data, detect anomalies or security breaches, and optimize cryptographic parameters for enhanced data protection. The ability of deep learning models

to discern complex relationships within encrypted information paves the way for more robust encryption schemes and efficient cryptanalysis, elevating data security to new heights.

3 Grain Cipher

The Grain cipher is a synchronous stream cipher that was introduced by Martin Hell and Thomas Johansson in 2003. Its design aims to provide a lightweight cryptographic primitive suitable for resource-constrained devices, such as RFID tags, which have limited memory and power capabilities. The cipher consists of three main building blocks: an LFSR (Linear Feedback Shift Register), an NFSR (Nonlinear Feedback Shift Register), and a filter function.

Grain cipher allows users to adjust the speed of the cipher based on the available hardware resources. This flexibility makes it suitable for resource-constrained devices, such as RFID tags, where efficiency is essential.

The key size in Grain cipher is 80 bits, and the Initialization Vector (IV) size is 64 bits. The cipher is designed to resist attacks, ensuring that no attack significantly faster than exhaustive key search can break it.

Grain cipher's strength and security have been extensively analyzed and compared to other well-known ciphers like E0 (used in Bluetooth) and A5/1 (used in GSM). Grain has demonstrated higher security while maintaining a small hardware implementation, making it a promising choice for various encryption applications, especially in resource-limited environments. Its balance of simplicity, security, and efficiency has positioned Grain cipher as a notable contribution to the field of stream ciphers.

3.1 Linear Feedback Shift Register (LFSR)

The LFSR is designed to produce a pseudo-random keystream and is governed by a feedback polynomial, f(x), of degree 80. This polynomial ensures a minimum keystream period and output balance. The LFSR in Grain cipher is an 80-bit register with feedback polynomial represented as:

$$f(x) = 1 + x^{18} + x^{29} + x^{42} + x^{57} + x^{67} + x^{80}$$

Additionally, the update function of the LFSR is designed to remove any possible ambiguity in the cipher's operation.The update function of the LFSR can be denoted as:

$$s_{i+80} = s_{i+62} \oplus s_{i+51} \oplus s_{i+38} \oplus s_{i+23} \oplus s_{i+13} \oplus s_i$$

where $s_i, s_{i+1}, ..., s_{i+79}$ represent the 80 bits of the LFSR at time i, and $\oplus$ denotes the bitwise XOR operation.

3.2 Non-Linear Feedback Shift Register (NFSR)

The NFSR, combined with a nonlinear filter, introduces nonlinearity to the cipher. The input to the NFSR is masked with the output of the LFSR to maintain balance in its state. Although referred to as an NFSR, it functions as a nonlinear filter, providing a higher level of security compared to traditional linear feedback shift registers.The NFSR in Grain cipher is an 80-bit register with a complex feedback polynomial, represented as follows:

$$g(x) = 1 + x^{17} + x^{20} + x^{28} + x^{35} + x^{43} + x^{47} + x^{52} + x^{59} + x^{65} + x^{71} + x^{80}$$
$$+x^{17}x^{20}+x^{43}x^{47}+x^{65}x^{71}+x^{20}x^{28}x^{35}+x^{47}x^{52}x^{59}+x^{17}x^{35}x^{52}x^{71}+x^{20}x^{28}x^{43}x^{47}$$
$$+x^{17}x^{20}x^{59}x^{65}+x^{17}x^{20}x^{28}x^{35}x^{43}+x^{47}x^{52}x^{59}x^{65}x^{71}+x^{28}x^{35}x^{43}x^{47}x^{52}+x^{59}x^{65}x^{71}x^{80}$$

The update function of the NFSR is denoted as:

$$b_{i+80} = s_i \oplus b_{i+63} \oplus b_{i+60} \oplus b_{i+52} \oplus b_{i+45} \oplus b_{i+37} \oplus b_{i+33} \oplus b_{i+28} \oplus b_{i+21}$$
$$+b_{i+15} \oplus b_{i+9} \oplus b_i \oplus b_{i+63}b_{i+60} \oplus b_{i+37}b_{i+33} \oplus b_{i+15}b_{i+9} \oplus b_{i+60}b_{i+52}b_{i+45}$$
$$+b_{i+33}b_{i+28}b_{i+21}\oplus b_{i+63}b_{i+45}b_{i+28}b_{i+9}\oplus b_{i+60}b_{i+52}b_{i+37}b_{i+33}\oplus b_{i+63}b_{i+60}b_{i+21}b_{i+15}$$
$$+b_{i+63}b_{i+60}b_{i+52}b_{i+45}b_{i+37}\oplus b_{i+33}b_{i+28}b_{i+21}b_{i+15}b_{i+9}\oplus b_{i+52}b_{i+45}b_{i+37}b_{i+33}b_{i+28}$$
$$\oplus b_{i+21}b_{i+15}b_{i+9}b_i$$

where $b_i, b_{i+1}, ..., b_{i+79}$ represent the 80 bits of the NFSR at time i.

3.3 Filter Function $h(x)$

The filter function $h(x)$ used in the Grain cipher is defined as follows:

$$h(x) = x_1 \oplus x_4 \oplus x_0x_3 \oplus x_2x_3 \oplus x_3x_4 \oplus x_0x_1x_2 \oplus x_0x_2x_3 \oplus x_0x_2x_4 \oplus x_1x_2x_4 \oplus x_2x_3x_4$$

where x_0, x_1, x_2, x_3, and x_4 correspond to the tap positions s_{i+3}, s_{i+25}, s_{i+46}, s_{i+64}, and b_{i+63} respectively. The function $h(x)$ combines these inputs using the bit-wise XOR operation ($\oplus$) to produce the output of the filter function.

The filter function $h(x)$ plays a crucial role in the keystream generation process of the Grain cipher, contributing to its high security level and resistance to cryptanalysis attacks.

3.4 Algorithm: Grain Cipher

3.4.1 Input:

- Key (K): 80-bit binary key
- Initialization Vector (IV): 64-bit binary IV
- Plaintext (PT): Binary plaintext to be encrypted

3.4.2 Output:

- Ciphertext (CT): Binary encrypted ciphertext

3.4.3 Steps:

1. Initialize the LFSR and NFSR states using the provided Key and IV.
2. Generate the keystream using the following loop:

```
for i = 1 to length of PT:
  a. Compute the filter function h(x) using inputs from e LFSR and NFSR.
  b. Calculate the output bit as the XOR of the least significant bit
     of the LFSR and the filter output.
  c. Shift the LFSR and NFSR to the right, updating their states.
  d. Store the output bit as a part of the keystream.
```

4 Data Generation

In this novel approach, we present an implementation of the Grain cipher, an efficient and secure stream cipher, using a combination of Linear Feedback Shift Registers (LFSRs) and Non-Linear Feedback Shift Registers (NFSRs). The Grain cipher is widely known for its simplicity and resilience against cryptographic attacks, making it an attractive choice for various encryption applications.

The core of our implementation revolves around the concept of generating pseudorandom keystreams from the given key and Initialization Vector (IV). We carefully construct the LFSR and NFSR states based on the input key, allowing for a robust and unpredictable keystream generation process. By leveraging the mathematical properties of the feedback polynomials, we achieve a balanced and highly nonlinear keystream, enhancing the security of the cipher.

To demonstrate the effectiveness of our approach, we utilize a random sample generation mechanism to produce a significant number of IVs and their corresponding keystreams. By employing a random seed based on the

current time, we ensure the uniqueness and randomness of the generated samples. Moreover, our method incorporates an essential step to eliminate duplicate entries, optimizing the dataset for further analysis.

Furthermore, we introduce a data transfer function that stores the generated IVs and keystreams in a CSV file. This enables seamless data retrieval and facilitates in-depth analysis to assess the randomness and security of the Grain cipher. The CSV file becomes a valuable resource for evaluating the cipher's performance under various scenarios and cryptographic scenarios.

Overall, our novel approach showcases the versatility and reliability of the Grain cipher as a stream cipher solution. By combining the power of LFSRs and NFSRs, our implementation ensures a high level of cryptographic strength, making it suitable for diverse encryption tasks in modern information security domains. Through our unique method of generating and analyzing keystreams, we contribute to the broader understanding of stream ciphers and their practical implications in real-world cryptographic applications.

4.1 Transfer Data to CSV

The `transferDataToCSV` function is responsible for transferring the generated IVs and keystreams to a CSV file named `data2.csv`. It ensures that each data entry is unique and avoids writing duplicates. The function takes in three parameters: `iv`, `keystream`, and `numSamples`.

Algorithm 1 Transfer Data to CSV

Open the `data2.csv` file in append mode
If the file fails to open, print an error message and return
Create a set of `DataEntry` structures to store unique data entries
Initialize `dataSize` to 0
{Write the column headers (currently commented out)}
{`fprintf(csvFile, "IV,Keystream n");`}
for $i = 0$ to `numSamples` **do**
    Convert binary IV and keystream arrays to hexadecimal strings using `snprintf`
    Store the hexadecimal IV in `ivStr` and the hexadecimal keystream in `keystreamStr`
    Create a new `DataEntry` struct with `ivStr` and `keystreamStr`
    if the `DataEntry` is not a duplicate (call `isDuplicate` function) **then**
        Write the IV to the CSV file followed by a comma
        Write the keystream to the CSV file
        Add the `DataEntry` to the set and increment `dataSize`
    end if
end for
Free the dynamically allocated memory for the `data` array
Close the CSV file
Print "Data transferred to data2.csv successfully."
=0

5 Machine Learning Model

The provided Python code employs a technical approach to construct a neural network model for predicting the first bit of the keystream in the Grain cipher. The initial step involves reading the data from a CSV file, which contains IV and keystream values represented in hexadecimal format. Subsequently, the code converts these hexadecimal values into binary format, facilitating further computational operations.

The dataset is partitioned into separate training and testing sets, a standard practice in machine learning for training and evaluating models effectively. The neural network architecture is constructed using the Keras Sequential API, comprising an input layer with 64 units to represent the 64-bit IV, a hidden layer with 32 units, and an output layer with 1 unit, utilizing the sigmoid activation function. For model compilation, the Adam optimizer and binary cross-entropy loss function are employed.

The training phase involves iteratively optimizing the model's internal parameters over the training set for a specified number of epochs, aiming to minimize the binary cross-entropy loss. Employing a batch size of 32 allows for efficient utilization of computational resources during model optimization.

Following training, the model's performance is assessed on the testing set, and the accuracy metric is computed. The achieved accuracy of approximately 50.3 percent indicates that the model's predictions exhibit only marginal improvement compared to random guessing for this particular task.

In a more technical sense, addressing the modest model accuracy necessitates further exploration and experimentation. This may involve investigating alternative neural network architectures, tuning hyperparameters, employing data augmentation techniques, or employing advanced optimization algorithms to enhance the model's predictive capabilities. Furthermore, expanding the dataset with more diverse samples could contribute to improved generalization and performance of the model. Overall, advancing the model's efficacy entails exploring various avenues to optimize its learning capacity and make more accurate predictions.

6 Experiments and Results

Hyper-Parameters	**Value**
Epochs	10
Data Size	2^{17} to 2^{18}
Train Size	75% of Data Size
Validation Size	19% of Train Size
Test Size	25% of Data Size
Loss Function	binary_crossentropy
Optimizer	Adam

Table 1: List of Hyper-Parameters

Bit	Batch Size	Epochs	Hidden Layers	Random State	Test Accuracy
1	32	10	3	45	0.5013
2	32	10	3	45	0.4976
3	32	10	3	45	0.5009
4	32	10	3	45	0.5024
5	32	10	3	45	0.5009
6	32	10	3	45	0.4977
7	32	10	3	45	0.4912
8	32	10	3	45	0.5004
9	32	10	3	45	0.5011
10	32	10	3	45	0.4995
11	32	10	3	45	0.4992
12	32	10	3	45	0.5016
13	32	10	3	45	0.4974
14	32	10	3	45	0.5007
15	32	10	3	45	0.5008
16	32	10	3	45	0.5001

(a) Experiment 1

Bit	Batch Size	Epochs	Hidden Layers	Random State	Test Accuracy
1	32	10	3	45	0.5013
2	32	10	3	45	0.4976
3	32	10	3	45	0.5009
4	32	10	3	45	0.5024
5	32	10	3	45	0.5009
6	32	10	3	45	0.4977
7	32	10	3	45	0.4912
8	32	10	3	45	0.5004
9	32	10	3	45	0.5011
10	32	10	3	45	0.4995
11	32	10	3	45	0.5013
12	32	10	3	45	0.5013
13	32	10	3	45	0.5013
14	32	10	3	45	0.5013
15	32	10	3	45	0.5013
16	32	10	3	45	0.5013

(b) Experiment 2

Table 2: Experiments Performed on Different Parameters

Bit	Batch Size	Epochs	Hidden Layers	Random State	Test Accuracy
1	32	10	3	45	0.5013
2	32	10	3	45	0.4976
3	32	10	3	45	0.5009
4	32	10	3	45	0.5024
5	32	10	3	45	0.5009
6	32	10	3	45	0.4977
7	32	10	3	45	0.4912
8	32	10	3	45	0.5004
9	32	10	3	45	0.5011
10	32	10	3	45	0.4995
11	32	10	3	45	0.4992
12	32	10	3	45	0.5016
13	32	10	3	45	0.4974
14	32	10	3	45	0.5007
15	32	10	3	45	0.5008
16	32	10	3	45	0.5001

(a) Experiment 3

Bit	Batch Size	Epochs	Hidden Layers	Random State	Test Accuracy
1	32	10	3	45	0.5013
2	32	10	3	45	0.4976
3	32	10	3	45	0.5009
4	32	10	3	45	0.5024
5	32	10	3	45	0.5009
6	32	10	3	45	0.4977
7	32	10	3	45	0.4912
8	32	10	3	45	0.5004
9	32	10	3	45	0.5011
10	32	10	3	45	0.4995
11	32	10	3	45	0.5013
12	32	10	3	45	0.5013
13	32	10	3	45	0.5013
14	32	10	3	45	0.5013
15	32	10	3	45	0.5013
16	32	10	3	45	0.5013

(b) Experiment 4

Table 3: Experiments Performed on Different Parameters

Bit	Batch Size	Epochs	Hidden Layers	Random State	Test Accuracy
1	32	10	3	45	0.5013
2	32	10	3	45	0.4976
3	32	10	3	45	0.5009
4	32	10	3	45	0.5024
5	32	10	3	45	0.5009
6	32	10	3	45	0.4977
7	32	10	3	45	0.4912
8	32	10	3	45	0.5004
9	32	10	3	45	0.5011
10	32	10	3	45	0.4995
11	32	10	3	45	0.4992
12	32	10	3	45	0.5016
13	32	10	3	45	0.4974
14	32	10	3	45	0.5007
15	32	10	3	45	0.5008
16	32	10	3	45	0.5001

(a) Experiment 5

Bit	Batch Size	Epochs	Hidden Layers	Random State	Test Accuracy
1	32	10	3	45	0.5013
2	32	10	3	45	0.4976
3	32	10	3	45	0.5009
4	32	10	3	45	0.5024
5	32	10	3	45	0.5009
6	32	10	3	45	0.4977
7	32	10	3	45	0.4912
8	32	10	3	45	0.5004
9	32	10	3	45	0.5011
10	32	10	3	45	0.4995
11	32	10	3	45	0.5013
12	32	10	3	45	0.5013
13	32	10	3	45	0.5013
14	32	10	3	45	0.5013
15	32	10	3	45	0.5013
16	32	10	3	45	0.5013

(b) Experiment 6

Table 4: Experiments Performed on Different Parameters

Round	Observations
3	The MLP1, MLP2, and MLP3 models are trained using training data consisting of 100,000 samples for S-values 1, 2, and 3 respectively. These models are then tested using testing data, achieving an average accuracy of 0.991. This accuracy indicates that the models/distinguisher can recognize certain patterns and successfully classify the dataset into two classes with a 0.99 accuracy. Since altering the number of hidden layers does not impact the results, we have decided to utilize only one hidden layer in the distinguisher.
4	The models MLP1, MLP2, and MLP3 are specifically designed for a data size of 2^{17} and S-values 1, 2, and 3, respectively. These models achieve an average testing accuracy of 0.786. While the accuracy in the fourth round differs from the third round, it remains above 0.5, indicating the non-random behavior of the PRESENT-80 cipher. Since the distinguisher accurately classifies the dataset for a data size of 2^{17}, we have decided not to conduct experiments for larger data sizes.
5	For data sizes ranging from 2^{17} to 2^{20} and S-values 1 and 2, the models MLP1, MLP2, and MLP4 achieve a testing accuracy of 0.5. Increasing the number of hidden layers in MLP4 and creating a new model MLP4(I) does not improve the accuracy. However, for the same data size and an S-value of 2, the accuracy of MLP2 starts to improve. In fact, for a data size of 2^{20}, MLP2 achieves a testing accuracy of 0.6. Similar trends are observed for S-value 3 as well. These observations suggest that, for a data size of 2^{20} and S-values 2 and 3, the distinguisher is able to identify certain patterns and exhibits non-random behavior.
6	In 6^{th} round, We have constructed two models for each set of data and S-value 1, 2 and 3, consisting of single hidden layer and two hidden layers. However, for data sizes ranging from 2^{17} to 2^{20} and S-values 1, 2, and 3, we observe an accuracy of 0.5. This indicates that the distinguisher is unable to classify the data set into two distinct classes up to that data size. It signifies the random behavior of the PRESENT-80 cipher in the 6^{th} round.
6	In 6^{th} round, We have constructed two models for each set of data and S-value 1, 2 and 3, consisting of single hidden layer and two hidden layers. However, for data sizes ranging from 2^{17} to 2^{20} and S-values 1, 2, and 3, we observe an accuracy of 0.5. This indicates that the distinguisher is unable to classify the data set into two distinct classes up to that data size. It signifies the random behavior of the PRESENT-80 cipher in the 6^{th} round.
6	In 6^{th} round, We have constructed two models for each set of data and S-value 1, 2 and 3, consisting of single hidden layer and two hidden layers. However, for data sizes ranging from 2^{17} to 2^{20} and S-values 1, 2, and 3, we observe an accuracy of 0.5. This indicates that the distinguisher is unable to classify the data set into two distinct classes up to that data size. It signifies the random behavior of the PRESENT-80 cipher in the 6^{th} round.

Table 5: Experimental Observations

7 Conclusion

References

[1] Author1. (Year1). Title of the first reference.

[2] Author2. (Year2). Title of the second reference.

[3] Author2. (Year2). Title of the second reference.

[4] Author2. (Year2). Title of the second reference.

[5] Author2. (Year2). Title of the second reference.

[6] Author2. (Year2). Title of the second reference.

[7] Author2. (Year2). Title of the second reference.

[8] Author2. (Year2). Title of the second reference.

[9] Author2. (Year2). Title of the second reference.

[10] Author2. (Year2). Title of the second reference.

[11] Author2. (Year2). Title of the second reference.

[12] Author2. (Year2). Title of the second reference.