Accelerating Training and Enhancing Security Through Message Size Optimization in Symmetric Cryptography

Abhisar[*], Madhav Yadav[**], Girish Mishra[***]

[*] Department of Mechanical Engineering, IIT Delhi, Delhi, India
abhisar.sos.07@gmail.com

[**] Department of Mathematics and Computing, IIT Hyderabad, Hyderabad, India
madhavyadav4595@gmail.com

[***] Scientific Analysis Group, Defence R&D Organisation, Delhi, India
gmishratech28@gmail.com

Abstract- This research extends Abadi and Andersen's exploration of neural networks using secret keys for information protection in multiagent systems. Focusing on enhancing confidentiality properties, we employ end-to-end adversarial training with neural networks Alice, Bob, and Eve. Unlike prior work limited to 64-bit messages, our study spans message sizes from 4 to 1024 bits, varying batch sizes and training steps. An innovative aspect involves training model Bob to approach a minimal error value close to zero and examining its effect on the feasibility of the model. This research unveils the neural networks' adaptability and scalability in encryption and decryption across diverse scenarios, offering valuable insights into their optimization potential for secure communication.

Keywords- *Symmetric neural network, Alice, Bob, Eve, Cryptography, Adversarial neural cryptography.*

1. INTRODUCTION

In recent years, the pervasive utilization of neural networks has remarkably advanced our capabilities in tackling increasingly intricate tasks. Notably, these networks have demonstrated their prowess in diverse applications, ranging from the deployment of Generative Adversarial Networks (GANs) [1] for generating and discerning fake images to our current exploration into the domain of symmetric Cryptography. Building on the seminal work of Martín Abadi and David G Andersen [2]., our research delves into the world of secure communication within a multiagent system [3], employing a sophisticated end-to-end adversarial training system featuring neural networks Alice, Bob, and Eve.

This investigation unfolds within the context of enhancing confidentiality properties, a critical pursuit in secure communication, especially when considering potential adversaries. Unlike traditional cryptographic approaches [4] that prescribe specific algorithms, our methodology embraces the flexibility of end-to-end adversarial training. This novel approach introduces an adaptive layer, allowing neural networks Alice and Bob to dynamically evolve in response to potential threats [5], particularly addressing information interception by a third neural network, Eve.

Expanding on the groundwork laid by Abadi and Andersen [2], our research pioneers comprehensive experiments with message sizes ranging from 4 to 1024 bits, in an effort to find the most optimized message size, while we also try various experiments to minimize our error and a lot more. This expansive exploration includes dynamic adjustments to batch sizes and training steps, providing an understanding of how neural networks adapt and scale in mastering encryption and decryption processes across a spectrum of message sizes and training conditions.
A unique key-point of our investigation involves the introduction of a novel dimension, where we try to train model Bob to approach its minimal error value, essentially converging towards zero. This objective not only refines the learning process but also emphasizes on maximizing the confidentiality goals achieved through the interactions of neural networks in our multiagent system.

Our paper is structured into five main sections. Section 1 is the introduction followed by Section 2 which provides a comprehensive overview of the model and its workings. In Section 3, we outline the experiments conducted on the model, and Section 4 presents the results and analyses. Section 5 is concerned with testing of the model. Finally, in Section 6, we draw conclusions from our research, emphasizing the potential community impact of our findings.

Each dimension of our research contributes unique benefits. The exploration of diverse message sizes enriches our understanding of neural network adaptability in different scenarios. Dynamic adjustments to batch sizes and training steps enhance the scalability and efficiency of the networks. The pursuit of minimizing the error of model Bob not only refines the learning process but also underscores our dedication to achieving the highest levels of confidentiality in secure communication. Overall, this research significantly advances our knowledge of the adaptability and optimization potential of neural networks, laying a solid foundation for future advancements in the realm of confidential information exchange.

2. UNDERSTANDING THE MODEL

2.1. OVERVIEW

The research simulates a cryptographic communication scenario involving three entities: Alice, Bob, and Eve. It initializes parameters and generates random shared keys and plain-text messages for Alice. A neural network architecture is defined [6][7][8], shared by Alice and Bob, along with a smaller one for Eve. The networks use convolutional layers and activation functions like sigmoid and tanh. The research implements loss functions to measure the separation between Alice's input and Bob's output, as well as the comparison of Eve's output to a random guess. Training involves alternating between training Alice/Bob and training Eve using an Adam optimizer. The losses for Bob and Eve are recorded and printed during training steps. Overall, the research explores the dynamics of secure communication, emphasizing the challenge of maintaining confidentiality in the presence of an eavesdropper.

2.2. MESSAGE AND KEY SIZE

In contrast to the methodology employed by Abadi and David, where key size and message size were consistently kept equal, our research introduces a deliberate variation in these sizes. Specifically, we explore sizes of 4, 8, 16, 32, 64, 128, 256, 512 and 1024 bits. This intentional diversification aims to understand the behavior of Bob and Eve when confronted with significantly larger key sizes. Throughout these experiments, the sample size is maintained as a constant value of 8192, allowing us to isolate and analyze the impact of key and message size variations on the performance of the neural networks involved. A sample size of 8192 was bifurcated into two parts, training and testing. The sample size was divided into batches, each containing 256 messages hence, a total of 32 batches: 25 for training and 7 for testing.

2.3. CREATION OF MESSAGE AND KEY

In this study, the 'message' and 'keys' are conceptualized as NumPy arrays, each having dimensions of (sample size, message size). To assemble these arrays, an encompassing structure of 8192 rows (sample size) was generated. Within each row, columns were created in accordance with the specified message size. Consequently, every row encapsulates a message with values set at either 1 or -1. These messages were randomly generated.

These randomly generated messages function as the plaintext received by Alice, destined for subsequent transformation into ciphertext. In parallel, a 2D array of keys was crafted, mirroring the number of rows at 8192 (sample size) and columns aligned with the bits in each key. This array symbolizes the shared key between Alice and Bob, a component for Bob to accurately predict ciphertexts.

2.4. LOSS FUNCTIONS FOR ALICE, BOB, AND EVE

The research incorporates a systematic approach to loss function design for the neural networks of Alice, Bob, and Eve [6]. The overarching objective is to guide Alice and Bob in minimizing their respective losses while concurrently ensuring that Eve's error consistently increases. Alice and Bob collaborate to enhance their performance, aiming for minimal loss, while Eve pursues the minimization of her error without access to the decryption key.

The design of these loss functions is pivotal in shaping the adversarial training process. The subsequent step involves setting these loss functions as minimization constraints, aligning with the collective goal of training each neural network to excel in its designated role.

2.5. BOB AND EVE LOSS FUNCTION

The algorithm leverages Hamming distance as a key metric to minimize errors. Hamming distance provides a quantitative measure of dissimilarity between binary sequences which in our case is the discrepancy of message generated by Bob and the original text. By systematically comparing the bits of the original and generated messages, the algorithm identifies discrepancies and aims to minimize the Hamming distance, effectively reducing errors in the output. This utilization of Hamming distance [9] serves as a fundamental component in the algorithm's strategy to enhance accuracy and optimize performance. This result is calculated for all the messages and averaged to give an average value of loss. In a similar way, the loss function for Eve was calculated.

2.6. COMBINED ALICE, BOB, EVE LOSS FUNCTION

In our neural network framework, the objective extends beyond minimizing Bob's loss; it also involves amplifying Eve's loss. To achieve this delicate balance, a combination of two functions is utilized: Bob's loss function and the difference between Eve's output and a random guess. Notably, the final result is not computed using Eve's loss function, but rather through assessing the difference between Eve's output and a random guess. This choice has been made to ensure that if Eve guesses all bits incorrectly, the network's overall dynamics remain relatively stable. This intentional insensitivity to complete incorrect guesses prevents possible hints that might aid Eve in deducing the encryption, reinforcing the goal of keeping Eve consistently unaware and unable to surpass random chance in her predictions. This approach ensures the careful consideration given to maintaining the integrity and security of the encryption process within the neural network architecture.

The Eve's distance from random guess is calculated as follows:

$$Eve\ Vs\ Random\ Guess = \frac{(Message\ Size - Eve\ Loss)^2}{Message\ Size^2}$$

A combined function was created such that it considers both the Bob's distance from original message and Eve's distance from random guess.

$$Alice\ Bob\ Eve\ Optimising\ Function = Bob\ Loss + Eve\ Vs\ RandomGuess$$

By minimizing this function, we can ensure that both Bob decrypts the message correctly while ensuring Eve cannot perform any better than a random guess.

2.7. NEURAL NETWORK

The neural network structure used by Bob, Eve, and Alice is as follows:

A function creates and initializes weight variables for the neural network layers. Another function takes an input, encrypt message (representing the encrypted message), and processes it through a series of operations.

It starts by performing matrix multiplication with a fully connected layer, followed by reshaping the result. Then, a sequence of convolutional layers is applied with varying filter sizes and strides, introducing non-linearities through activation functions like sigmoid and tanh. These convolutional layers contribute to the network's ability to learn features from the input data. The final result is reshaped to match the expected output size.

2.8. TRAINING OF ALICE, BOB AND EVE

The neural network-based symmetric encryption system involves distinct objectives for its participants. Alice, responsible for encrypting plaintext, aims to create a cipher decipherable by Bob while thwarting Eve's attempts to accurately guess the content. Bob, tasked with decryption, endeavors to accurately recover the original plaintext given the ciphertext and the key. Meanwhile, Eve's objective is to decipher the message solely based on the ciphertext. To achieve these goals, the system employs the Alice Bob Eve Optimising Function, which minimizes the combined loss of Alice and Bob during training. Implemented with the Adam optimizer, the training process monitors the loss function for both Bob and Eve after each iteration. Collaboration among the participants ensures the encryption and decryption processes are harmoniously aligned to meet the desired objectives. Additionally, the training regimen focuses on minimizing Eve's loss, thereby enhancing her chances of decrypting the code correctly. The termination conditions for the code include reaching a predetermined number of steps or achieving an optimal value for Bob's loss, with each condition tailored as required by the experiment.

2.9. EXTRACTING DATA

The data was extracted from the model in one of the three ways:

1. By fixing the number of steps, for example the model would stop operating after executing 5,000 steps.

2. By fixing the Loss of Bob. By constraining the lower limit of loss for Bob, we were able to calculate the number of steps required to reach that accuracy. Please note that we will be interchangeably using the terms Bob loss and error which mean the same thing: how far Bob’s decryption of the ciphertext is from the original plain text.

3. By changing batch size. The effect of batch size on the loss value was observed keeping the value of bits constant.

The next section discusses the experiments conducted by us on the model and our methodology behind them.

3. EXPERIMENTS ON OUR MODEL

3.1. MESSAGE SIZE V/S NUMBER OF EPOCHS FOR FIXED ERROR

This experiment focuses on investigating the impact of varying

Message sizes and the associated number of epochs required for training our model. Throughout the experiment, a constant error value was set, specifically 0.5 and 0.1 in our case. The study involved conducting trials with Message sizes of 4 to 1024 bits, and recording the corresponding number of epochs needed to achieve a predetermined error level.

The objective of this experiment is to identify the optimal message size, aiming to minimize the number of iterations required for effective training. This research holds potential applications [3][11] in various fields where efficient training processes are crucial for developing models with improved speed and resource utilization.

3.2. SAMPLE SIZE V/S ERROR AT CONSTANT EPOCH

In the experiment conducted by Anderson and David [2], the sample size was consistently set at 4096. However, our current investigation seeks to assess the impact of different sample sizes on the results. In this context, the sample size refers to the number of plaintexts and keys that traverse our training model during each epoch. By systematically varying the sample size, we aim to gain insights into how this parameter influences the final outcomes of the training process. However, for maintaining a uniformity, the batch size was kept constant to 256 messages. We tried experimenting with a sample size of 2048, 4096 and 8192 messages. Each of these samples were divided into batches of size 256 messages. This exploration is essential for a more nuanced understanding of the relationship between batch size and the achieved results, shedding light on potential optimizations, how to achieve desired accuracy with minimal training time.

3.3. MESSAGE SIZE V/S ERROR FOR A FIXED NUMBER OF EPOCHS

The focus of this experiment is to investigate how errors change under a constant number of training cycles, specifically set at 5,000 epochs to ensure consistency. In a methodical manner, we altered the bit size throughout the experimentation process and observed the corresponding errors. By systematically adjusting the bit size and measuring errors at each step, we aim to gain insights into how different bit sizes impact error metrics. This structured approach allows us to analyze and understand the relationship between varying bit sizes and the resulting error rates, providing valuable information about the sensitivity of our model to changes in bit configuration.

3.4. NUMBER OF EPOCHS V/S ERROR FOR FIXED MESSAGE SIZE

This experiment is mainly focused on understanding how the number of training cycles, known as epochs, affects reaching a particular level of error in our model. We're keeping it simple by creating a graph that shows the relationship between the number of epochs and the error. This graph helps us make predictions about how many epochs are needed to achieve a specific level of accuracy. In other words, we're trying to figure out the best number of training cycles to get our model to perform as accurately as we want it to.

4. RESULTS AND ANALYSIS

4.1. RESULTS: MESSAGE SIZE V/S NUMBER OF EPOCHS FOR FIXED ERROR

In the course of this experiment, our algorithm was subjected to predefined termination criteria based on a specific Loss Value. The experiment encompassed four distinct scenarios where the designated loss values were set at 1, 0.75, 0.5, and 0.25. It was our hypothesis that analyzing the behavior of the algorithm under these diverse loss values would yield valuable insights, guiding us in understanding the feasibility and effectiveness of our algorithm in achieving the desired loss targets.

The results have been shown below (Fig 1., Fig 2. & Fig 3.)

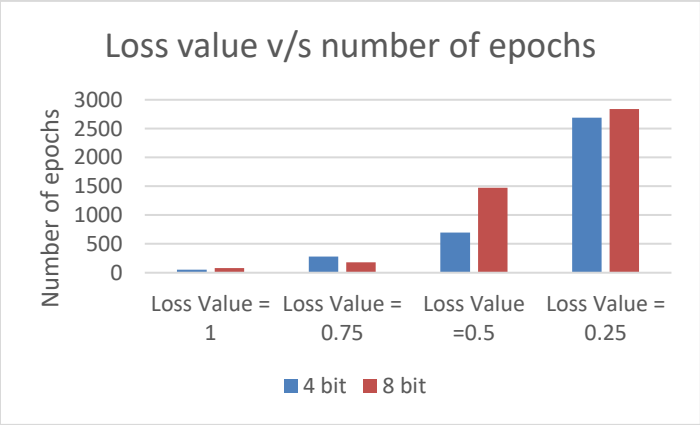

Fig 1. Bar graphs for 4 bit and 8 bit message

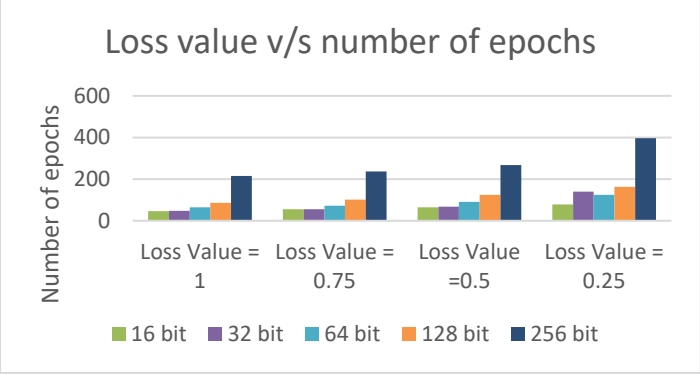

Fig 2. Bar graphs for 16, 32, 64 and 256 bit message

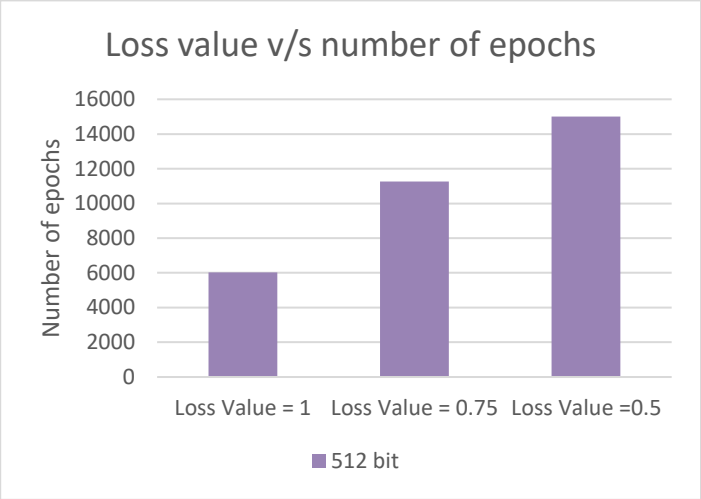

Fig 3. Bar graphs for 512 bit message

Observations from the two plotted graphs, where the first had a constant error of 0.5 and the second a constant error of 0.1:

1. As can be seen, the scale for 4 bit, 8 bit was totally different to that of 16, 32, 64, and 256 bit messages which was again different from 512 bit message. This result implies that it is much easier for messages larger than 8 bits to quickly reach their optimal value (minimum loss value) than 4 and 8 bit messages.

2. For higher message sizes, more epochs were needed to compensate for the increased number of possibilities. Between the sizes of 16 to 64, an optimal message size was identified. Within this range, the message size was sufficient for the algorithm to function effectively without being excessively large, resulting in a reduced need for epochs.

3. In general, as lesser loss value was desired, the number of epochs rose for every bit. For message sizes of 512 bits the loss value of 0.25 was not achieved even after 25,000 epochs and remained stagnant to minimal value of 0.4. A similar observation was observed for 1024 bits where even after 15,000 epochs, a minimal loss value of 30 was observed.

4. The observed pattern between 16 and 64 suggests a sweet spot for the most optimal message size during the model training process. This range allows for efficient algorithm performance without requiring an excessive number of epochs.

4.2. RESULTS: SAMPLE SIZE V/S ERROR AT CONSTANT EPOCH

The primary objective of this experiment was to investigate the extent to which we could reduce the sample size, leading to a decrease in computational power requirements without significantly compromising the error rate. The initial setup involved a sample size of 8192 messages, organized into batches

of 256 messages each. The sample size was systematically reduced, reaching a minimum of 2048 bits. Beyond this point, any further reduction in the sample size markedly diminished accuracy on the test data.

To assess the impact of these variations, we compared Bob's loss values on both the training and testing sets upon the completion of 5000 epochs. The ensuing observations are detailed below.

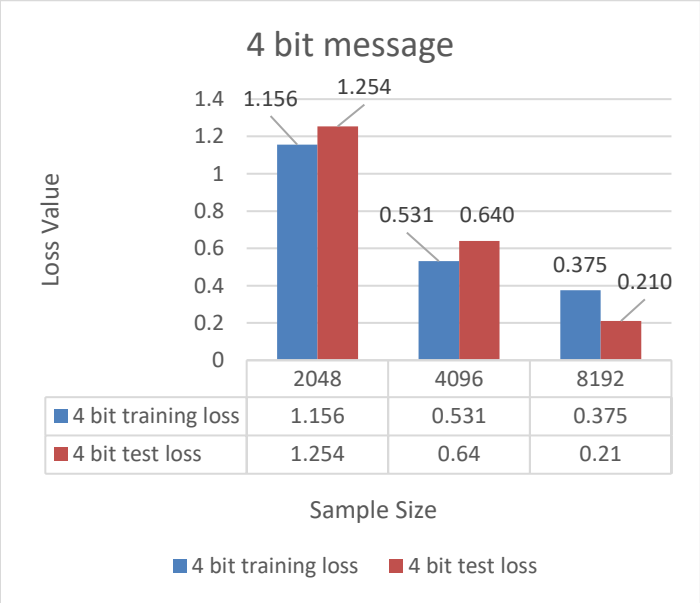

Fig 4. Bar graphs for different sample sizes, 4 bit message

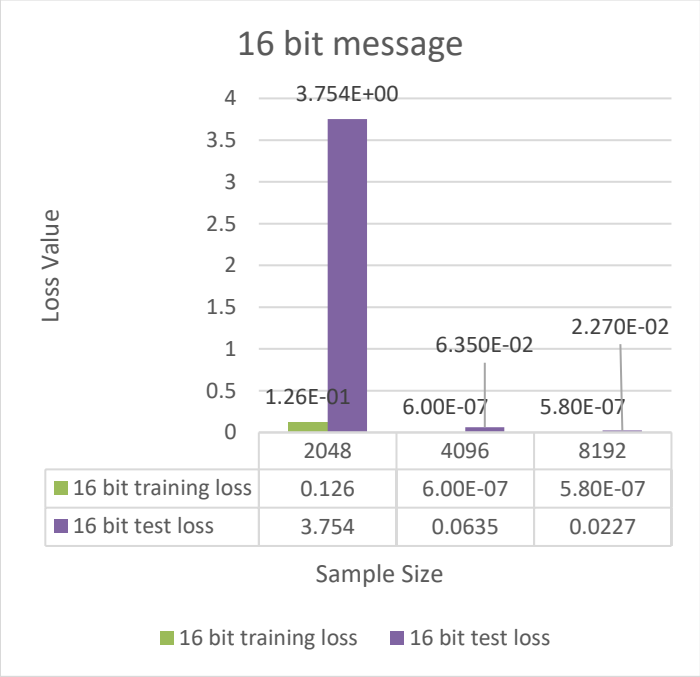

Fig 5. Bar graphs for different sample sizes, 16 bit message

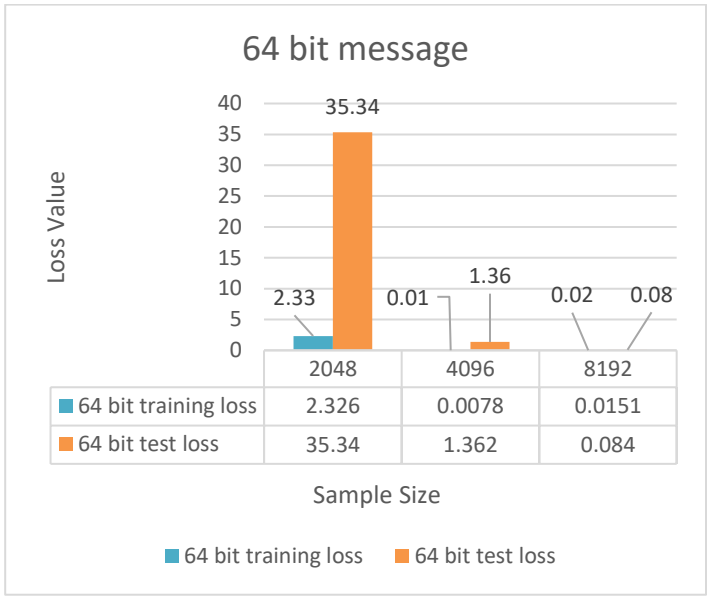

Fig 6. Bar graphs for different sample sizes, 64 bit message

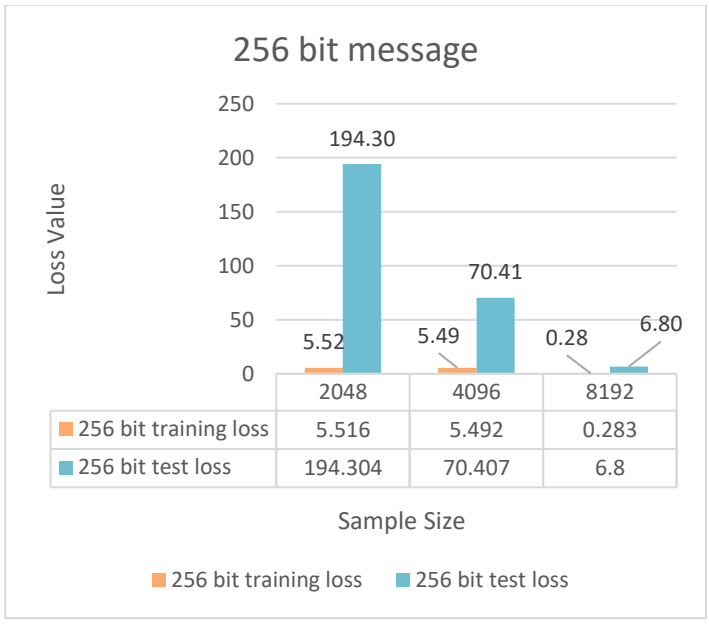

Fig 7. Bar graphs for different sample sizes, 256 bit message

The results can be summarized as follows:

1. In the case of 4-bit messages (Fig 4.), the training and testing errors exhibit some comparability, potentially stemming from the minimal training requirements for small-sized messages. However, in line with our prior experiments, 4-bit messages still manifest a notably high error rate in both the training and testing datasets.

2. In contrast, across all other scenarios, a substantial discrepancy in training and testing errors is evident for varying sample sizes. Notably, the 8192-sample size yields significantly lower errors compared to the 2048 and 4096 sample sizes.

3. Observations reveal an upward trend in error rates as we increase the message size while maintaining a constant sample size.

4. Interestingly, the 16-bit message size emerged as a favorable choice, demonstrating negligible errors in both the training and testing datasets.

4.3. RESULTS: MESSAGE SIZE V/S ERROR FOR A FIXED NUMBER OF EPOCHS

In this study, we maintained a consistent number of epochs at 5000, selecting this value based on a balance between achieving a desirable error reduction and ensuring computational feasibility. The choice of 5000 epochs was motivated by their sufficiency in minimizing the error to 0.05, aligning with our experimental objectives. Importantly, this epoch configuration allowed for the successful execution of our experiments without imposing excessive computational demands. Going beyond this threshold to achieve higher accuracy levels would have necessitated a significant increase in computational time, a trade-off that was deemed unnecessary given the satisfactory results obtained at the chosen epoch size.

The test was done for message sizes of 4, 8, 16, 32, 64, 128, 256, and 1024 bits. The error at the end of 5000 epochs for each of these cases was noted and the results were observed.

The images below show the variation of error v/s epoch for different bits (4, 8, 16, 32, 64, 128, 256, and 1024)

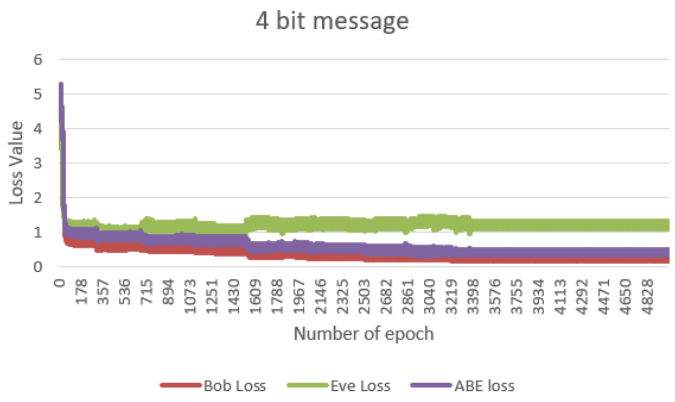

Fig 8. Epoch number v/s Loss value for 4 bit message

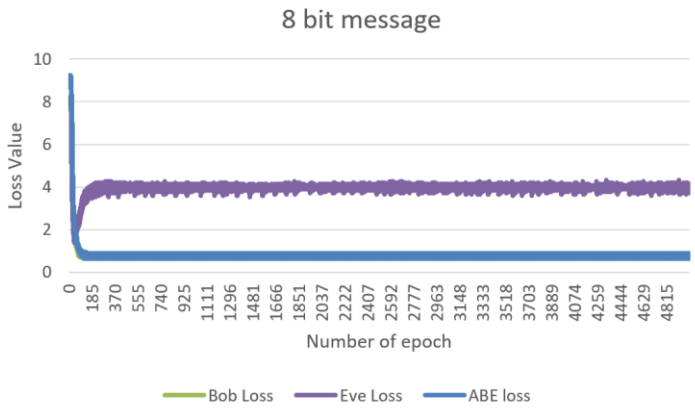

Fig 9. Epoch number v/s Loss value for 8 bit message

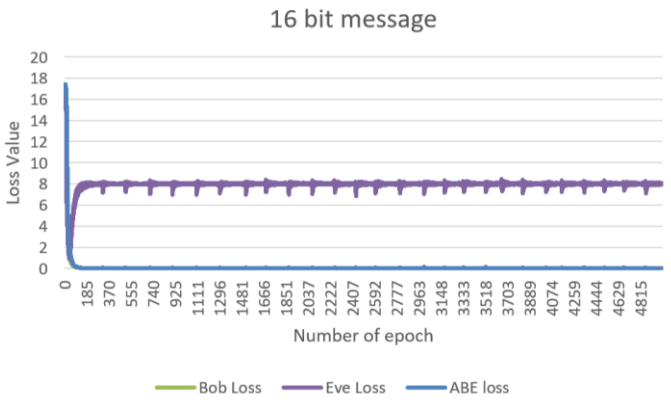

Fig 10. Epoch number v/s Loss value for 16 bit message

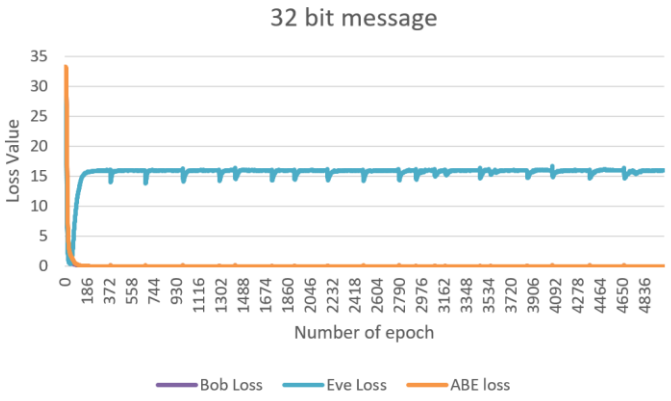

Fig 11. Epoch number v/s Loss value for 32 bit message

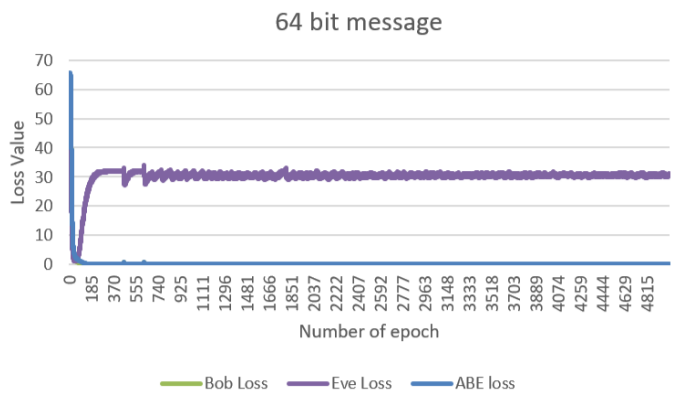

Fig 12. Epoch number v/s Loss value for 64 bit message

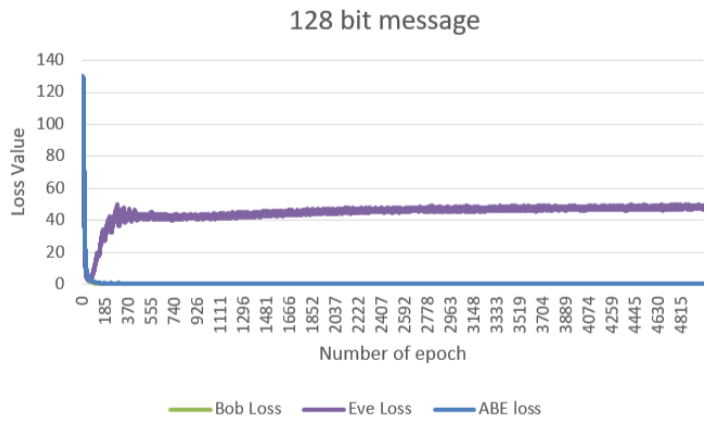

Fig 13. Epoch number v/s Loss value for 128 bit message

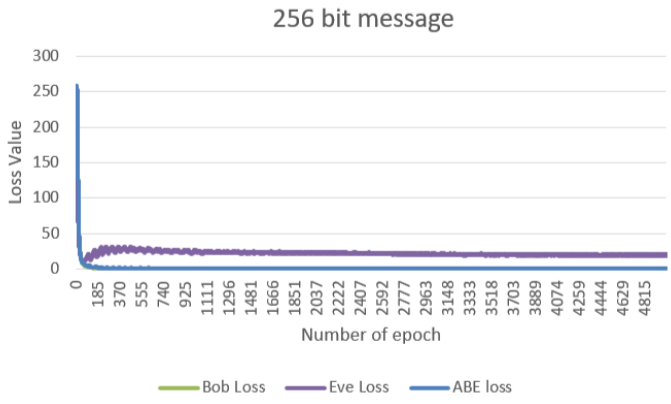

Fig 14. Epoch number v/s Loss value for 256 bit message

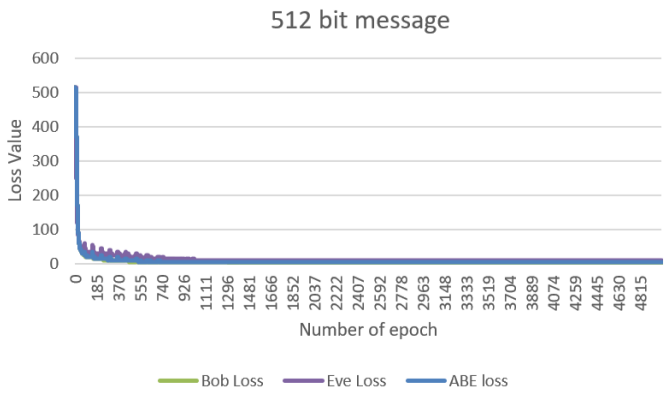

Fig 15. Epoch number v/s Loss value for 512 bit message

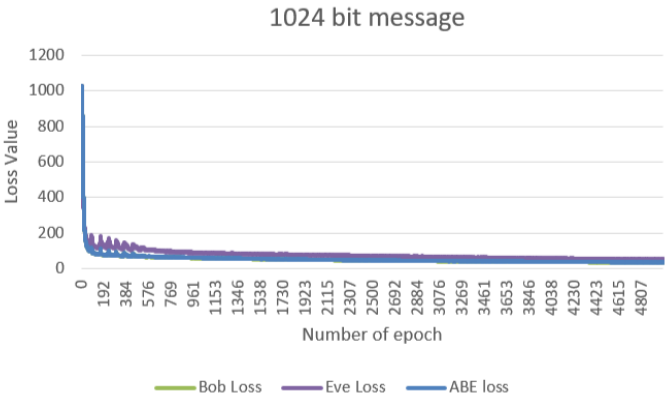

Fig 16. Epoch number v/s Loss value for 1024 bit message

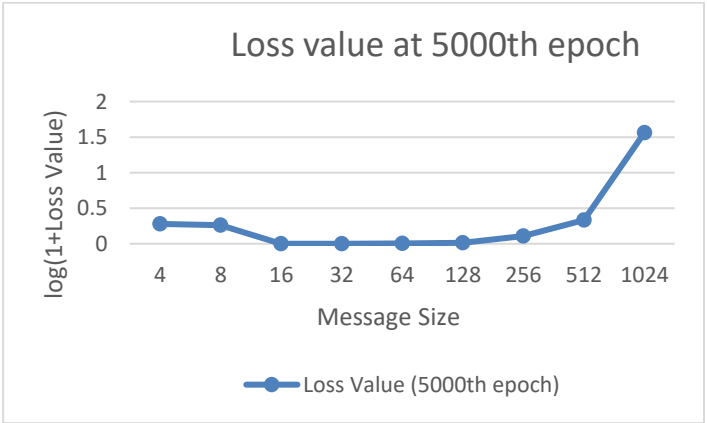

Fig 17. Error v/s bit size at 5000th epoch (log_{10})

Upon analyzing the graphs, the following observations were made:

1. For a 4 bit message (Fig 8.), the graph is very noisy. The noisiness decreases as the message as the message size increases.

2. From 16 bit message to 64 bit message (Fig 10, 11, 12), the graphs have significantly less amount of noise while keeping Eve's error ideal. This however changes as we increase the message size, implying on increasing message size, both Bob and Eve notice a significant decrease in their losses nearly becoming equal as the message size increases. (Fig 14. to Fig 16). This result is of very importance because it shows that for larger message sizes, if eve is trained enough the security of the ciphertext could be compromised.

3. The minimum recorded error, observed after 5000 epochs, was for the 32-bit message (Fig 11.), specifically 4.2468×10^{-7}. Conversely, the maximum error was recorded for the 1024-bit message (Fig 16.), amounting to 35.41.

4. The error remained relatively stable for message sizes up to 256 bits but exhibited an exponential surge thereafter. The error recorded for a 256-bit message (Fig 14.) was 0.283. A plausible explanation for this pattern could be that once the algorithm surpasses a certain error threshold, (0.5 in our case), further reduction becomes notably challenging. Consequently, a substantial decrease in error was not observed.

5. The number of steps necessary to achieve a specific error value increases with an increase in message size.

6. The largest message size with correspondingly sufficient minimization of error after 5000 epochs (Fig 17.) was 64 bit size and 16 bit. This might be because as the size increases, the possibilities for our model to amend changes also increases, hence increasing the number of epochs required.

4.4. RESULTS: NUMBER OF EPOCHS V/S ERROR FOR FIXED MESSAGE SIZE

In this experiment, the focus shifted to plotting the number of epochs against the loss value. The objective was to observe the variation in the number of epochs required from the beginning to the end for each error value. This experiment is designed to provide insights into the dynamic relationship between the number of epochs and loss values, offering a basis for predicting an approximate estimate of the epochs needed to reach a specific loss value for a given message size and error threshold.

Consider the following graphs:

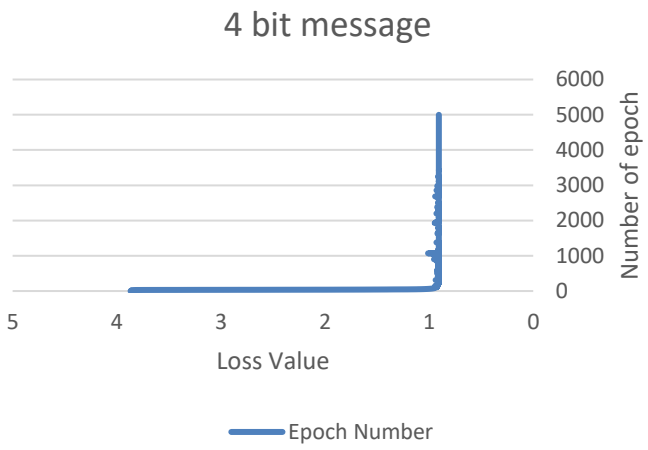

Fig 18. Loss value v/s Epoch number for 4 bit message

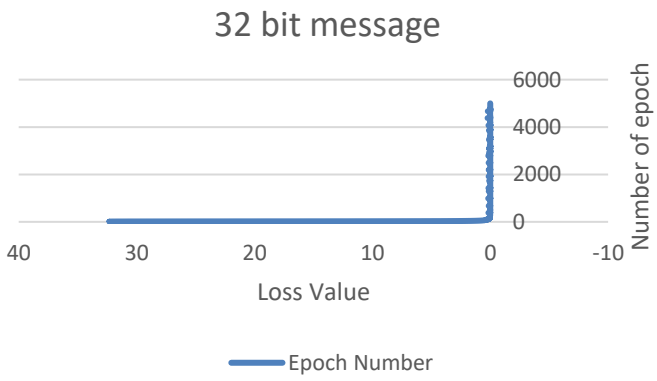

Fig 19. Loss value v/s Epoch number for 32 bit message

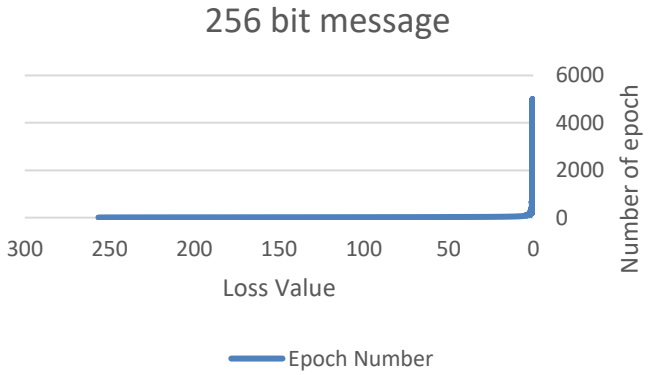

Fig 20. Loss value v/s Epoch number for 256 bit message

The following things can be observed here:

1. A scatter plot was employed to illustrate the density of points for a specific Bob's loss value. Notably, the graphs revealed a significantly higher density as the error value approached zero. This heightened density near zero suggests a concentrated convergence behavior around this threshold.

2. Furthermore, a notable surge in the number of epochs was observed when the error value approached zero. This observation aligns with our earlier finding that crossing a certain threshold value prompts a substantial increase in the number of epochs required to minimize the error. The sudden jump in epochs near zero underscores the algorithm's increased challenge in further reducing error beyond this critical point.

Let us consider a special case where we sample size = 4096 and the whole sample gets trained together. This was done to reduce the noise for this particular experiment. For a 256 bit message the graph was observed as follows:

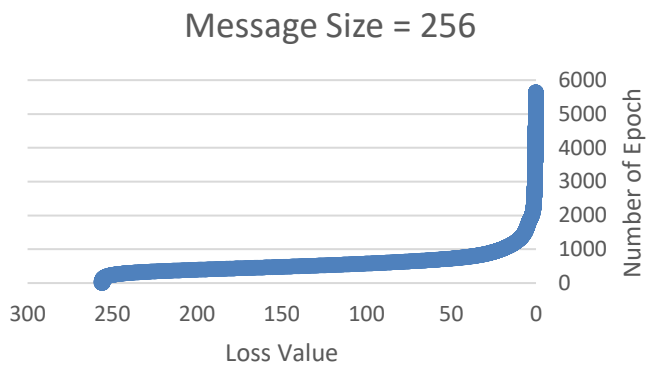

Fig 21. Loss value v/s Epoch number for 256 bit message with 4096 samples

By observing these trends, a formula was devised to predict possible number of epochs for a given error:

$$\text{NumberOfEpoch} = -a\ln(LossValue) + b$$

Where a and b are positive real numbers.

By taking few initial values of number of epoch and error, a and b can be found.

This was tested for 128 bit message where the predicted function came out to be $y = -559.6ln(x) + 2645.7$ and corresponding R^2 value came out to be 0.9559.

This was repeated for 256-bit message, where equation came out to be $y = -661ln(x) + 3163.2$ and R^2 value came out to be 0.9078 and for 32 bit message $y = -1066ln(x) + 2609.6$ and R^2 value of 0.8972.

This result could prove to be quite effective in determining whether the accuracy required is feasible or not. Since the function for 256 bit message is known, we can easily find a rough estimate on the number of times we have to train our model so that the model becomes accurate.

However, it should be noted that this is just a rough approximation and is limited to this case only. It can be used to find an approximate order of epochs required and not its exact value.

Also, the accuracy of this varies with number of points considered for finding a and b. In general, more the number and spread out the points were, more the accuracy of this function.

5. TESTING OF OUR MODEL

The preceding sections concentrated on scrutinizing the impact of varying parameters on the algorithm during training. No test data was employed, and the model underwent continuous training on a consistent number of samples. However, the objective of this section is to assess our algorithm and determine the average loss value. This assessment will ultimately enable us to draw conclusions regarding the credibility of our model.

Analyzing the comparison of testing Bob loss with other parameters reveals several trends (Table 1).

1. For a given epoch, the training loss value of Bob and the testing loss value of Bob were closely aligned up to 8 bits. The divergence between training and testing loss values starts to occur from 16 bits onwards.

2. Up to 128 bits, the loss value is consistently below 1, with the lowest loss values in the testing set recorded for 16-bit messages.

3. From 256 bits onwards, a notable increase in Bob's loss is observed.

4. Up to 64 bits, an increase in the number of epochs did not significantly impact testing accuracy. This suggests that the models were not prone to overtraining.

5. A consistent trend indicates that lower loss values during training correlate with lower loss values during testing.

6. While the gap between loss values of Bob in testing and training datasets wasn't substantial, the same cannot be said for Eve. Eve exhibited very high error values for higher message sizes, even when their training losses were significantly lower.

A lot more results can be observed from another experiment we did in the previous section (section 4.2)

1. We observed there was a significant drop in bob loss for tests as the sample sizes were increased across all the cases.

2. The lowest loss value of bob in while testing was observed for 16 bit messages when comparing two bits having same number of epochs.

message size	Number of epoch	Training set loss value (Bob)	Training set loss value (Eve)	Testing set loss value (Bob)	Testing set loss value (Eve)
4	500	0.517	0.648	0.540	1.243
4	5000	0.375	1.203	0.210	1.203
8	500	0.821	4.127	0.801	4.062
8	5000	0.820	4.094	0.797	3.934
16	500	9.738E-04	7.496	0.017	7.602
16	5000	5.844E-07	7.980	0.023	8.474
32	500	6.892E-04	15.985	0.054	16.365
32	5000	4.247E-07	15.969	0.080	16.661
64	500	1.483E-03	30.896	0.322	32.323
64	5000	1.542E-05	30.228	0.085	32.809
128	5000	0.107	47.340	1.086	50.496
128	10000	0.047	49.363	0.666	55.362
256	5000	0.284	19.710	6.800	35.373
256	10000	0.231	22.350	6.221	37.190
512	5000	1.105	8.625	39.577	52.640
512	10000	0.704	5.713	36.741	50.418
1024	5000	36.673	51.571	435.223	293.203
1024	10000	25.906	36.099	418.483	307.173

Table 1: Effect of number of epochs and batch size on testing Bob loss

message size	Training set loss value (Bob)	Training set loss value (Eve)	Testing set loss value (Bob)	Testing set loss value (Eve)
16	0.1	7.142	0.133	7.133
32	0.1	14.727	0.268	13.887
64	0.1	28.034	0.980	27.888
128	0.1	33.401	3.398	36.498
256	0.1	23.911	11.140	43.975

Table 2: Testing Bob loss when training Bob loss is fixed

On observing the Table 2 , the loss value during training for bob was fixed to 0.1 and the corresponding results were observed.

1. Consistent with our previous findings, it was once again observed that the loss value for Bob during testing was minimum for 16-bit messages, followed by 32-bit messages. It's worth noting that 4-bit, 8-bit, and 512-bit messages were unable to reduce their Bob loss to 0.1; therefore, they have been omitted from the table. It should be noted that even eve showed significant improvement in its accuracy,

especially for 256 bit message size where she was able to guess most of the bits correctly.

6. CONCLUSION

The primary purpose of undertaking this research was to gain a comprehensive understanding of the dynamics of symmetric cryptography, with a specific focus on the impact of varying parameters on algorithmic performance during training and testing phases. Through a series of experiments, we tried to unravel the nuanced relationships between Message size, error rates, number of epochs, and batch sizes, aiming to optimize the functioning of cryptographic algorithms.

In the course of our extensive experiments, we have gleaned valuable insights that culminate in significant conclusions regarding the performance and optimization of our symmetric cryptography algorithm. Each experiment contributed unique perspectives to our understanding.

Firstly, in the exploration of varying epochs versus message size for fixed error, we identified a crucial factor for achieving desired accuracy—the most optimal message size. Our findings indicate that, for our algorithm, a message size ranging from 16 to 64 bits emerges as the most favourable for attaining the desired level of accuracy.

Moving on to the investigation of batch size versus epoch at constant error, we sought to determine the minimum batch size that balances computational efficiency with result quality. Our conclusive observation is clear: a sample size below 4096 compromises accuracy, while larger sample sizes, around 4096 or 8192, yield superior results. This knowledge provides a practical guideline for optimizing the batch size in the algorithm.

In the examination of varying epochs and observing loss value, we discerned that 16 to 64-bit messages consistently outperformed other message sizes, including smaller sizes like 4 and 8 bits. These mid-range message sizes exhibited minimal noise and ideal values for Eve's loss, further emphasizing their efficacy in the algorithm.

The final experiment delved into the challenging task of minimizing Bob's loss value after reaching a certain threshold. Our observation of an exponential increase in the number of steps required beyond a minimum loss value led us to construct a predictive function. While offering a rough estimate, this function serves as a valuable tool for understanding the computational demands of further error reduction.

Concluding with the testing of our model, we affirm the credibility of our algorithm across various parameters. In summary, our comprehensive analysis allows us to assert three key conclusions: 1) A message size ranging from 16 to 64 bits proves to be the most optimum for our algorithm. 2) A minimum sample size of 4096 is imperative for achieving lower loss values. 3) While reducing error to a single-digit number is achievable for most cases, the exponential increase in the required number of epochs for further reduction poses a noteworthy challenge.

Collectively, this research significantly contributes to the field of symmetric cryptography by unraveling key insights and optimizing algorithmic performance. The paper offers a foundation for refining cryptographic algorithms and understanding the interplay between parameters, providing practical implications for real-world applications [3][11]. Future research endeavors can build upon these findings, exploring additional parameters, refining predictive models, and extending the application of symmetric cryptography to address evolving challenges in the realm of cybersecurity. As we look ahead, this work lays the groundwork for a more nuanced and efficient approach to cryptographic algorithm design and implementation.

7. REFERENCES

[1] Goodfellow, I. J. (2014, June 10). *Generative adversarial networks*. arXiv.org. https://arxiv.org/abs/1406.2661

[2] Abadi, M. (2016, October 21). Learning to Protect Communications with Adversarial Neural Cryptography. arXiv.org. https://arxiv.org/abs/1610.06918

[3] I. Meraouche, S. Dutta, S. K. Mohanty, I. Agudo and K. Sakurai, "Learning Multi-Party Adversarial Encryption and Its Application to Secret Sharing," in IEEE Access, vol. 10, pp. 121329-121339, 2022, doi: 10.1109/ACCESS.2022.3223430.

[4] Meraouche, Ishak & DUTTA, Sabyasachi & Tan, Haowen & Sakurai, Kouichi. (2021). Neural Networks Based Cryptography: A Survey. IEEE Access. PP. 1-1. 10.1109/ACCESS.2021.3109635.

[5] William Stallings. 2010. Cryptography and Network Security: Principles and Practice (5th. ed.). Prentice Hall Press, USA.

[6] *Adversarial neural cryptography(October 20, 2018) - the Mathy Bit*. (n.d.). https://mathybit.github.io/adversarial-neural-crypto/

[7] Nuclearstar. (n.d.). *Adversarial_Neural_Cryptography/main.py at master · Nuclearstar/Adversarial_Neural_Cryptography*. GitHub. https://github.com/Nuclearstar/Adversarial_Neural_Cryptography/blob/master/Research_Paper.pdf

[8] VamshikShetty. (n.d.). *adversarial-neural-cryptography-tensorflow/train.py at master · VamshikShetty/adversarial-neural-cryptography-tensorflow*. GitHub. https://github.com/VamshikShetty/adversarial-neural-cryptography-tensorflow/blob/master/train.py

[9] Bookstein, Abraham & Kulyukin, Vladimir & Raita, Timo. (2002). Generalized Hamming Distance. Information Retrieval. 5. 10.1023/A:1020499411651.

[10] Chandra, Sourabh & Bhattacharyya, Siddhartha & Paira, Smita & Alam, Sk. (2014). A Study and Analysis on Symmetric Cryptography. 10.1109/ICSEMR.2014.7043664.

[11] Sooksatra, K., & Rivas, P. (2020). A Review of Machine Learning and Cryptography Applications. *2020 International Conference on Computational Science and Computational Intelligence (CSCI)*, 591-597.