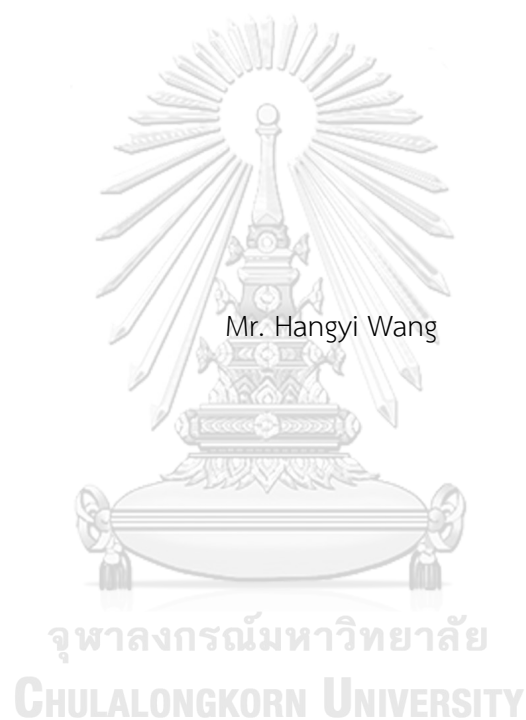


CRYPTOSYSTEM FOR TEXTUAL MESSAGE BY USING MALLEABLE NEURAL NETWORK
WITH SECRET DIMENSIONS



A Thesis Submitted in Partial Fulfillment of the Requirements
for the Degree of Master of Science in Computer Science and Information Technology
Department of Mathematics and Computer Science
FACULTY OF SCIENCE
Chulalongkorn University
Academic Year 2020
Copyright of Chulalongkorn University

ระบบเข้าและถอดรหัสสำหรับข้อความอักษรโดยใช้เครือข่ายประสาทที่ปรับโครงสร้างได้ร่วมกับมิติลับ



วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิทยาศาสตรมหาบัณฑิต
สาขาวิชาวิทยาการคอมพิวเตอร์และเทคโนโลยีสารสนเทศ ภาควิชาคณิตศาสตร์และวิทยาการ

คอมพิวเตอร์

คณะวิทยาศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย

ปีการศึกษา 2563

ลิขสิทธิ์ของจุฬาลงกรณ์มหาวิทยาลัย

Thesis Title	CRYPTOSYSTEM FOR TEXTUAL MESSAGE BY USING MALLEABLE NEURAL NETWORK WITH SECRET DIMENSIONS
By	Mr. Hangyi Wang
Field of Study	Computer Science and Information Technology
Thesis Advisor	Professor CHIDCHANOK LURSINSAP, Ph.D.

Accepted by the FACULTY OF SCIENCE, Chulalongkorn University in Partial
Fulfillment of the Requirement for the Master of Science

----- Dean of the FACULTY OF SCIENCE
(Professor POLKIT SANGVANICH, Ph.D.)

THESIS COMMITTEE

----- Chairman
(Associate Professor SUPHAKANT PHIMOLTARES, Ph.D.)

----- Thesis Advisor
(Professor CHIDCHANOK LURSINSAP, Ph.D.)

----- External Examiner
(Prem Junsawang, Ph.D.)

จุฬาลงกรณ์มหาวิทยาลัย
CHULALONGKORN UNIVERSITY

ทางหยี หวัง : ระบบเข้ารหัสและถอดรหัสสำหรับข้อความอักษรโดยใช้เครือข่ายประสาทที่ปรับ
โครงสร้างได้ร่วมกับมิติลับ. (CRYPTOSYSTEM FOR TEXTUAL MESSAGE BY USING
MALLEABLE NEURAL NETWORK WITH SECRET DIMENSIONS) อ.ที่ปรึกษาหลัก : ศ. ดร.
ชิตชนก เหลือสินทรัพย์

การเข้ารหัสในปัจจุบันส่วนใหญ่ใช้การคำนวณแบบโมดูลอด้วยกฎแฉาธารณะและ
ส่วนตัวที่กำหนดไว้ล่วงหน้า ความแข็งแกร่งของแนวทางนี้ถูกควบคุมโดยขนาดของตัวแบ่งหลัก
กระบวนการเข้ารหัสและถอดรหัสของการประมวลผลแบบโมดูลอนี้ค่อนข้างเร็วและค่อนข้าง
แข็งแกร่งสำหรับการโจมตีใด ๆ อย่างไรก็ตาม ยังคงเป็นไปได้ที่จะถอดรหัสข้อความที่เข้ารหัสโดยใช้
อัลกอริธึม meta-heuristic เครื่องคู่ขนานที่มีประสิทธิภาพสูง และแนวคิดการคำนวณควอนตัม
วิทยานิพนธ์นี้ใช้แนวทางอื่นเพื่อเพิ่มความทนทานของข้อความที่เข้ารหัสโดยปรับใช้โครงสร้างของ
โครงข่ายประสาทเทียมแบบ feedforward ที่มีมิติข้อมูลลับเสริม เห็นด้วยทั้งผู้ส่งและผู้รับ
โครงสร้างเครือข่ายสามารถปรับให้เดาได้ยาก แนวคิดที่นำเสนอแตกต่างจากการชิงโครโนชันน้ำหนัก
และการแสดงน้ำหนักของข้อความที่เข้ารหัส ข้อความตัวอักษรเป็นปัญหาหลัก ของการศึกษาครั้งนี้
อย่างไรก็ตาม สามารถขยายแนวคิดที่เสนอเพื่อรับมือกับรูปแบบอื่น ๆ ของข้อความอินพุท ผลการ
ทดลองแสดงให้เห็นถึงความแข็งแกร่งของข้อความที่เข้ารหัสเพื่อต่อต้านการโจมตีโดยใช้
cryptanalysis กล่าวถึงความซับซ้อนของพื้นที่และเวลาของเครือข่ายและกระบวนการ



สาขาวิชา	วิทยาการคอมพิวเตอร์และ เทคโนโลยีสารสนเทศ	ลายมือชื่อนิสิต
ปีการศึกษา	2563	ลายมือชื่อ อ.ที่ปรึกษาหลัก

6172629423 : MAJOR COMPUTER SCIENCE AND INFORMATION TECHNOLOGY

KEYWORD: Cryptosystem, Cryptography, Neural cryptography, Artificial neural network, Multi-Layered Neural Network, Information Technology

Hangyi Wang : CRYPTOSYSTEM FOR TEXTUAL MESSAGE BY USING MALLEABLE NEURAL NETWORK WITH SECRET DIMENSIONS. Advisor: Prof. CHIDCHANOK LURSINSAP, Ph.D.

Most of the current cryptography is based on modulo computing with predefined public and private keys. The robustness of this approach is controlled by the size of the prime divider. The encryption and decryption processes of this modulo computing are rather fast and rather robust to any attack. However, it is still possible to crack the encrypted message by using meta-heuristic algorithm, very high performance parallel machines, and also quantum computing concept. This thesis takes another approach in order to enhance the robustness of encrypted messages by deploying the structure of a feedforward neural network with augmented secret dimensions concurred by both sender and receiver. The network structure can be adjusted to make it difficult to guess. The proposed concept differs from weight synchronization and weight representation of the encrypted message. The text message is the main concern of this study. However, the proposed concept can be extended to cope with other forms of input messages. The experimental results showed the robustness of encrypted messages against attacks by using cryptanalysis. The space and time complexities of the network and process are discussed.

Field of Study: Computer Science and
Information Technology

Student's Signature

Academic Year: 2020

Advisor's Signature

ACKNOWLEDGEMENTS

I would like to express my gratitude to all those who have helped me in the way to complete this thesis.

First, I am grateful to my advisor, Professor Chidchanok Lursinsap who directs, suggests, and helps me all the time during my research. While my advisor is busy and has a lot of students to direct, he still discusses with me the process of my thesis every two weeks and helps me to solve some hard questions which I met for the first time in the life.

Secondly, I would like to show my thanks to the rest of my thesis committee: Associate Professor Suphakant Phimoltares and Dr.Prem Junsawang from the Department of Statistics, Faculty of Science, Khon Kaen University. Associate Professor Suphakant Phimoltares gave me a lot of help with the format of writing my thesis. And all the committee devoted the time to read and gave helpful comments on my thesis to make it better.

Thirdly, Thanks to Ms. Nonglak Pootachareon, as the teacher assistant, who gave me a lot of help with my questions about the format of the thesis.

Especially, I want to express my deep love to my family and friends for their unselfish love and encouragement. They have supported me along the way.

จุฬาลงกรณ์มหาวิทยาลัย
CHULALONGKORN UNIVERSITY

Hangyi Wang

TABLE OF CONTENTS

	Page
ABSTRACT (THAI)	iii
ABSTRACT (ENGLISH)	iv
ACKNOWLEDGEMENTS	v
TABLE OF CONTENTS	vi
LIST OF TABLES	9
LIST OF FIGURES.....	10
CHAPTER 1 INTRODUCTION	11
CHAPTER 2 BACKGROUND.....	13
2.1 Traditional cryptography.....	13
2.2 Other models.....	17
CHAPTER 3 PROPOSED METHOD.....	19
3.1 Our Method	19
3.2 Difference with other models.....	23
3.3 Model Design.....	24
3.3.1 Scale the data	25
3.3.2 Running Time.....	26
3.3.3 Message Length change	26
3.3.4 Encryption And Decryption key.....	27
3.3.5 Fail Rate Of Decryption.....	27
3.3.6 The Secret Key	28
CHAPTER 4 EXPERIMENTS	30

4.1 Data set.....	30
4.2 Encryption part	31
4.3 Decryption part.....	36
4.4 Testing environment.....	37
4.5 Training strategies.....	38
4.5.1 Weights and biases initialization	38
4.5.2 Input size	39
4.5.3 Learning rate	40
4.5.4 Loss function	40
4.5.5 Optimization function.....	41
4.5.6 Early stopping.....	41
4.5.7 Callback.....	42
4.5.8 TensorBoard.....	42
4.5.9 Device strategy.....	47
4.6 Why choosing TensorFlow.....	51
4.7 Save and load models.....	52
4.8 Maintenance cryptosystem.....	52
CHAPTER 5 ANALYSIS OF OUR MODEL	54
5.1 Complexity analysis.....	54
5.2 Cryptanalysis.....	55
5.2.1 Normal attack.....	55
5.2.2 Cipher text only attack.....	55
5.2.3 Chosen plain text attack.....	62
5.2.3.1 Number of cipher nodes less than number of plain text nodes...64	

5.2.3.2 Number of cipher nodes larger than number of plain text nodes	65
5.2.3.3 Number of cipher nodes equal to number of plain text nodes	66
CHAPTER 6 SUGGESTION AND CONCLUSION	68
6.1 Suggestion of setting cryptosystem	68
6.2 Suggestion when cryptosystem fails	70
6.3 Conclusion	70
Appendix I SAMPLE CHAPTER	71
A.1 Sample	71
Appendix II LIST OF PUBLICATIONS	72
B.1 International Conference	72
VITA	75



LIST OF TABLES

	Page
Table 2.1 The pros and cons of typical Cryptography Algorithms.	4
Table 2.2 The minimum key length of different cryptography algorithms.	6
Table 4.1 Encryption and decryption time of three examples.	26
Table 5.1 Three examples of network structure.	47
Table 5.2: Attack result.	47
Table 5.3: Detail of the examples of three situations.	53
Table 5.4: Comparative of DES, AES, RSA with our method of encryption time, decryption time and cryptanalysis time.	57



LIST OF FIGURES

	Page
Figure 3.1 Proposed concept	10
Figure 3.2 Structure of our model.	14
Figure 4.3 One sample in the samples directory	20
Figure 4.4 Dataset	21
Figure 4.5 Loss vs number of hidden neurons for “Hello, World!”	22
Figure 4.6: Loss vs number of hidden neurons and secret key for message one....	23
Figure 4.7 Loss vs secret key for “Hello, World!”	24
Figure 4.8 Loss vs hidden neurons and secret key for “What do you want to eat in the evening?”	25
Figure 4.9 Loss, hidden neurons and secret key for How about we eat in the same restaurant we ate yesterday evening?	26
Figure 4.10 Network structure to encrypt and decrypt three example messages....	27
Figure 4.11 TensorBoard of one model of our cryptosystem.	34
Figure 4.12 The GRAPHS tab of TensorBoard.	35
Figure 4.13 The GRAPH of architecture of the model on TensorBoard.	36
Figure 4.14 Time Series tab on TensorBoard.	36
Figure 4.15 GPU MirroredStrategy: Supports Synchronous Distributed Training With Multiple GPUs On Single Machine.	38
Figure 4.16 TPU Strategy Our team used in the lab	39
Figure 4.17 Output of the TPU after using the TPU strategy in the lab.	40
Figure 5.1 The python files we used in local computer.....	49
Figure 5.2 Hacking the Message 1 and guessing from nodes 2 to 13.	50
Figure 5.3 Hacking the Message 1 and guessing from nodes 14 to 20.	51
Figure 5.4 2-14 Hack Result.	55
Figure 5.5 20-5 Hack Result.	55
Figure 5.6 10-10 Hack Result.	56
Figure 6.1 Reference when setting the cryptosystem from our experiments.	58

CHAPTER 1

INTRODUCTION

With the development of meta-heuristic algorithm, high performance parallel machines, distributed computing and quantum computing [1], the complexity of cracking the encrypted communication is not as secure as before. The complexity of cracking the encrypted communication is controlled by the time spent to guess the keys in today's asymmetric cryptography algorithms, which are mostly based on modulus functions. As a result, a new cryptography method for messages ought to be more sophisticated than current approaches such as utilizing a neural network or an elliptic curve [2] to make the cracking steps harder. Besides, the RSA cryptosystem normally used is still secure nowadays but its steps of encryption and decryption are inflexible.

Based on the above, a new cryptography method is proposed by deploying one explicit joint conic neural structure. and a adaptable single secret key with multiple blocks varying lengths. With our cryptography method, the sender could encrypt all kinds of textual messages with different lengths, and the length of the cipher could be easily varied as well. Recently, various neural cryptography attempts have been described. To build efficient private key [3], several researchers synchronized the networks by using twin time-dependent weights. The effects of this was also used by the dynamics of neural cryptography [4], Nonetheless, those methods focused solely on key exchange schemes without flexible realization of non-linear functions depend on neural network.

Neural cryptography is divided into three types: (1) multi-layered neural networks, (2) synchronized neural networks, and (3) chaotic neural networks.

Some researchers have already used a neural network to study cryptography and shown that cracking the encrypted communication is difficult [5], following that,

the Google Brain researchers trained a new system that used a generative adversarial network without attempting to learn any specific techniques [6], Nonetheless, this schema is difficult to examine and deploy since it did not provide any information about what the algorithm studied [7]. While this schema proved to be robust [8]. The researchers in [9], used multi-layered neural networks to realize the generative adversarial network. The encoder and decoder of this method performs admirably, while the security needs to be enhanced.

In terms of space and time complexity, this paper uses a multi-layered neural network technique with a special joint conic structure and secret key to strengthen the hardness of guessing the neural network structure and weights. Integers are used as the input in our cryptosystem since they are easy to check and alter for beginners who are unfamiliar with the cryptosystem. Integers could also be replaced by float or binary numbers. The binary number is used as the input for neural networks in other cryptosystems.

The following sections made up the rest of this paper:

The background of classical cryptography and other models are briefly summarized in Section 2. The proposed method is discussed in Section 3. The experiment of our method with detail are shown in Section 4. The complexity and security are analyzed and examined in Section 5. And some advice about the cryptosystem and conclusion are given in the last Section.

CHAPTER 2

BACKGROUND

2.1 Traditional cryptography

People have used the transposition and substitution ciphers for message confidentiality in the beginning [7, 10]. For example, Caesar cipher is one of the most famous substitution ciphers which replaced each letter in the plaintext with the other letter several fixed number of slots further down the alphabet table.

Two significant techniques used in encryption are secret key (symmetric) and public-key (asymmetric) encryption, classified based on the number of keys. Commonly, asymmetric encryption uses one public key known by everyone for encryption and another private key utilized for decryption. But for symmetric encryption, two individuals share the same key to code and decode the information. There exists another method typically implementing a digital fingerprint named Hash functions which uses a mathematical conversion to encrypt the message irreversibly.

With the development of cryptography, there are also several common ways developed by the people used to crack the cipher, such as the brute force attack. The brute-force attack method only works for the cipher that could be calculated, and the possibility is finite. For English alphabet, since there is only 26 lower alphabets. Some character is used more often than other characters. For example, "e" is the most common alphabet in the communication of normal people. Hence, the frequency and distribution analysis is another powerful tool used to crack the cipher with only English alphabets and symbols. Another common way used to attack the cipher is the dictionary attack, which used a common cipher in one dictionary to compare with the cipher being hacked. While the dictionary will be very large, and it is very possible not to find any correct cipher in the end.

With the popular of machine learning, people have used it in the field of cryptography. In 2000, John Kelsey, Bruce Schneier, et al. using side channel to do the cryptanalysis towards product ciphers [11], And in 2019, Liyao Xiang, Haotian Ma, et al. proposed using the complex-valued neural network in this field as well to protect privacy information [9].

Table 2.1 The pros and cons of typical Cryptography Algorithms.

Cipher	Pros and cons
1. Reverse cipher	easy to operate and hack
2. Substitution cipher	
2.1 The simple substitution cipher	effectively invulnerable; the weakness is frequency and distributions of symbols
2.2 Caser cipher	more possible keys; while only all 66 possible keys; mathematical proven unbreakable while hard to practice
2.4 Vigenere cipher	invincible to the word pattern attack; the weakness is repeating of the key
2.5 DES	use a Feistel network while not secure now
2.6 AES	high speed and low RAM requirements; a variant of Rijndael; while uses simple algebraic structure and encrypt every block the same way
3. Transposition cipher	much more keys while still can use brute-force to hack
4. Asymmetric Cryptography	much more computationally intensive than symmetric ones; in theory, weak to a "brute-force key search attack."
5. Hash Functions	well-suited for ensuring data integrity; while they are not easily decipherable

Including the common hack methods, the table **2.1** briefly shows the pros and cons of traditional encryption and decryption algorithms. Since there are too many algorithms for encryption and decryption, we just select the typical cryptography methods in the table according to their develop time.

In the beginning of Table **2.1**, Reverse cipher may be the easiest method that only reverses the message to encode and decode. Additionally, there are two main branches of Cryptography, distinguishes as substitution and transposition cryptography [10]. The substitution cipher replaces units of plain text according to the set-up rules. The simple substitution cipher, Caesar cipher, Affine cipher, and Vigenère cipher are all typical examples of substitution ciphers. Furthermore, there is one quite particular type of substitution cipher called the one-time pad, which is simple and yet only mathematical proven unbreakable. One-time pad is a stream cipher if the key meets these criteria:

1. it is truly random.
2. its length must be at least as long as the plaintext.
3. it is not reused in any part of the message.
4. it must be kept completely secret.

Vigenère cipher is a poly-alphabetic substitution cipher that can be taken as the two-time pad cipher. A brute-force dictionary attack and more advanced techniques such as the **Kasiski examination** and **Friedman test** can be used to find the prime weakness of the Vigenère cipher. It is easily broken after the attacker finds the key length. Compared to the substitution cryptography superseding a plain text with other symbols, the transposition cryptography readjusts the position of the letters in the plain text. The substitution and transposition ciphers may be combined to gain the name of product cipher.

In symmetric encryption, two individuals share the same key while two keys are normally used in asymmetric encryption, one for encryption and the other for decryption. The number of bits is employed to measure the robustness of the cryptography key. Table 2.2 summarizes the number of bits in each cipher [12].

Table 2.2 The minimum key length of different cryptography algorithms.

Type	Usage	Minimum Key Length (bits)
Symmetric key	Personal	128
	Small Business	192
	Big Company	256
Discrete key	Personal	250
	Business	250

The key length is quite different for various cryptography, and the minimum length for the same algorithm also needs to be changed with the development of mathematics and hardware. Generally, the weakness in cryptography is fundamentally based upon key management rather than weak keys [11], and the suggestion of key size could only be a minimal setting reference around 2021. In our cryptosystem, the cipher key is the weights and biases of the input layer to the hidden layer if there is only three layers in the model, which means it is easy to arrive the minimal secure length that needs to be used under different conditions.

2.2 Other models

There are three basic models: Restricted Boltzmann Machine (RBM), Principal component analysis (PCA) and Autoencoder.

RBM only has 2 layers, the first layer is for input and second layer is for output. In more detail, RBM only train 2 hidden nodes in the hidden layer and send out either 0 or 1.

RBM has two phases: Forward pass and Backward pass (or reconstruction).

The output of the hidden units is the probability distribution. In other words, RBM uses input data to make predictions about the probability of output based on the given weights.

PCA is a method using the most important components to perform a change of basis on the input, which mainly used for dimensionality reduction.

As for Autoencoder, it is one kind of artificial neural networks which reproduces the input and improves the Restricted Boltzmann Machine (RBM). The difference between autoencoder and RBM is that the former has 3 layers and the latter has 2 layers [13]. In other words, the output of one autoencoder is the same as the input. Compared with our model, the autoencoder also has the encoder layer, coder layer, decoder layer. The main usage for autoencoder is also dimensionality reduction, except this, image denoising and image compression are the applications that autoencoder also performs well [14]. About data separation, autoencoder provided us with a better result comparing with PCA.

The autoencoder has several common variations: Sparse autoencoder (SAE), Concrete autoencoder, Variational autoencoder (VAE). On the code layer, the sparse autoencoder involves a sparsity penalty to the training criterion. The sparsity penalty helps model to activate specific areas of the neural network on the basis of the input data while inactivating all other neurons.

While our model compared with the autoencoder, we have the secret dimension as the secret key to the input layer. The secret key could not only be the hash value to check the consistency of the information, but also with plasticity dimensions could help to resize the input data.



CHAPTER 3

PROPOSED METHOD

3.1 Our Method

Figure **3.1** illustrates the proposed concept. Only a three-layered network topology consisting of is explored to simplify the proposed method's explanation.

The input contains the message to be encrypted, a secret key dealt by the cryptosystem will be added to the input in the input layer of the neural network. The cryptosystem would continue to train the network until the outputs match the input (loss less than 0.5) automatically.

In the beginning, the weight and bias matrix between the input layer and hidden layer, as well as the secret key, are utilized as keys to encode the message, which are then dealt by the cryptosystem to produce the cipher. To decode the cipher and verify it, the receiver uses the upper part of the same network structure to get the original message along with the secret key to verify no third person alter the cipher. The secret key including the number of secret dimensions and accurate value, structure of the network consisting of number of hidden neurons and input neurons, and number of layers could be set by the cryptosystem automatically or decided by the sender and receiver.

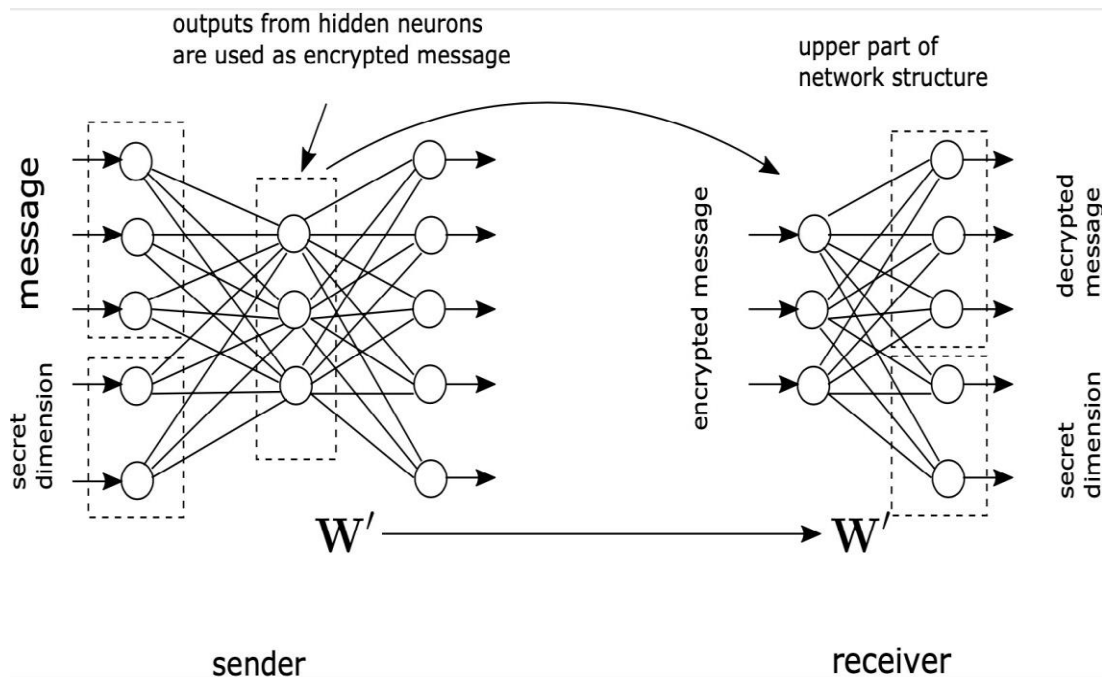


Figure 3.1 Proposed concept

The first layer including (1) a set of input neurons which represents the message $x = \{x_1, x_2, \dots, x_M \mid x_i \in R\}$; (2) a secret key $s = \{s_1, s_2, \dots, s_B \mid s_i \in R\}$.

The cryptosystem would randomly allocate the physical positions of M and S in the input layer. In more detail, the first n_1 neurons may be allocated for M , whereas the remaining are allocated for S . Normally, number of layers may be varied freely depending on the encryption message's resilience and security.

The second layer is made up of a group of nodes $H = \{h_1, h_2, \dots, h_C\}$, C could be larger, equal or smaller than $M + B$. As a result, the cipher size may be smaller, equal to, or bigger than the plaintext. The number of neurons in third layer is totally the same as in first.

Both the message $y = \{y_1, y_2, \dots, y_M \mid y_i \in R\}$ and the secret key $s' = \{s'_1, s'_2, \dots, s'_B \mid s'_i \in R\}$ are produced by the output neurons, and $Y = X$, $S' = S$.

Use $w_i^{(hidden)}$, and $b_i^{(hidden)}$ to represent the weight, bias matrix of hidden neurons h_i .

So every input pattern is a matrix represented as XS . The following is how h_i it is calculated.

$$h_i = g(XS \cdot w_i^{(hidden)} + b_i^{(hidden)})$$

In our Python application, the default activation function $g(\cdot)$ was Rectified Linear Unit (*Leaky-Relu*) which could also be changed to others.

The value of each $w_i^{(hidden)}$ and $b_i^{(hidden)}$ are generated by training the network with the Adam [15] algorithm. We employ the adaptive learning rate in our model instead of the schedule learning rate in our structure. All $w_i^{(hidden)}$ are placed in the set W .

The third layer will produce 2 sets of neurons. The first set Y generates the same message as X . The weight and bias matrices of output neurons y_i in Y are represented as $w_i^{(out)}$ and $b_i^{(out)}$. The second set S' generates the same output as S . The weight and bias matrices of secret-dimension neurons s'_i are represented as $w_i^{(secret)}$ and $b_i^{(secret)}$.

For each output, the neuron utilizes the result of the dot product of the weights and inputs, and that result is adjusted according to the biases. The integer outputs of the network must be rounded to have the same value as the inputs. Each output neuron's y_i and s_i could be calculated as follows.

$$y_i = \begin{cases} \lceil g(h_i \cdot w_i^{(out)} + b_i^{(out)}) \rceil & \text{if } g(\cdot) \geq 0.5 \\ \lfloor g(h_i \cdot w_i^{(out)} + b_i^{(out)}) \rfloor & \text{if } g(\cdot) < 0.5 \end{cases}$$

$$s'_i = \begin{cases} \lceil g(h_i \cdot w_i^{(secret)} + b_i^{(secret)}) \rceil & \text{if } g(\cdot) \geq 0.5 \\ \lfloor g(h_i \cdot w_i^{(secret)} + b_i^{(secret)}) \rfloor & \text{if } g(\cdot) < 0.5 \end{cases}$$

The cipher includes $w_i^{(out)}$, $w_i^{(secret)}$, $b_i^{(out)}$, $b_i^{(secret)}$, and \mathbf{i} . The W' and b' are cipher part.

The processes for our method are shown in Algorithm 1 and Algorithm 2.

Algorithm 1: Encryption

Inputs: (1) a secret key represented by S . (2) the message values represented by X . (3) the number of hidden neurons if the sender want to set.

Output: (1) $w_i^{(out)}$, $w_i^{(secret)}$, $b_i^{(out)}$, and $b_i^{(secret)}$ for all \mathbf{i} . (2) the cipher H .

The network structure information including the number of layers and number of neurons in different layers.

Algorithm 2: Decryption

Inputs: (1) cipher H . (2) one weight matrix W' and one bias matrix b' . (3) a secret key represented by S . (3) the information about network structure including the layer count and neuron count in different layers.

Output: decrypted message.

1. Use the upper neural network part for decryption.
2. The weights and biases matrix represented by W' and b' automatically calculated by the cryptosystem using the received information from the sender.
3. Feed H and calculate output $\mathbf{O} = \{Y, s'\}$.
4. Extract the secret key S' , Y from \mathbf{O} , if one attacker has altered the cipher, then the secret key would also be affected and different with the original one.
5. Check If $S = S'$, $Y = X$, then the cipher has not been altered.

3.2 Difference with other models

Compared with other models, our cryptosystem is more flexible and adjustable according to different status. There are several differences between our cryptosystem and other approaches in these aspects, e.g. the model design, normalization of the data, running time, the length of the message change, the encryption and decryption key, the fail rate to decode the cipher and the secret key used in our model. The model design is the prototype of the cryptosystem which could also be added more hidden layers. Not only the secret dimensions, but also the hidden layer in our neural cryptosystem has plasticity.

If the model has many layers, the encrypted message could be taken from the output of any layer. For example, except the input and output layer, one neural model has six hidden layers, the first three hidden layers used for encryption and the later three layers used for decryption part. Then, the cipher of the cryptosystem could be produced by the any layer of the first three hidden layers. Normally, we set the last hidden layer for the encryption part as the output of the cipher. Technically, any hidden layer including the hidden layer used for the decryption could also be used for generating the cipher.

The structure of our model is shown in Figure 3.2. The upper part of the cryptosystem is for the sender to set and adjust the configuration. The latter part of the cryptosystem is used for the receiver to adjust and set the configuration to obtain the decrypted message.

In our model, it is clear that the encryption key and decryption key are distinguish. It is convenient to let the cryptosystem to generate a new unique key if one loses the encryption key by using a random seed number. The structure of the network can be adjusted according to this key to speed up the convergence of encryption process based on synaptic weight adjusting.

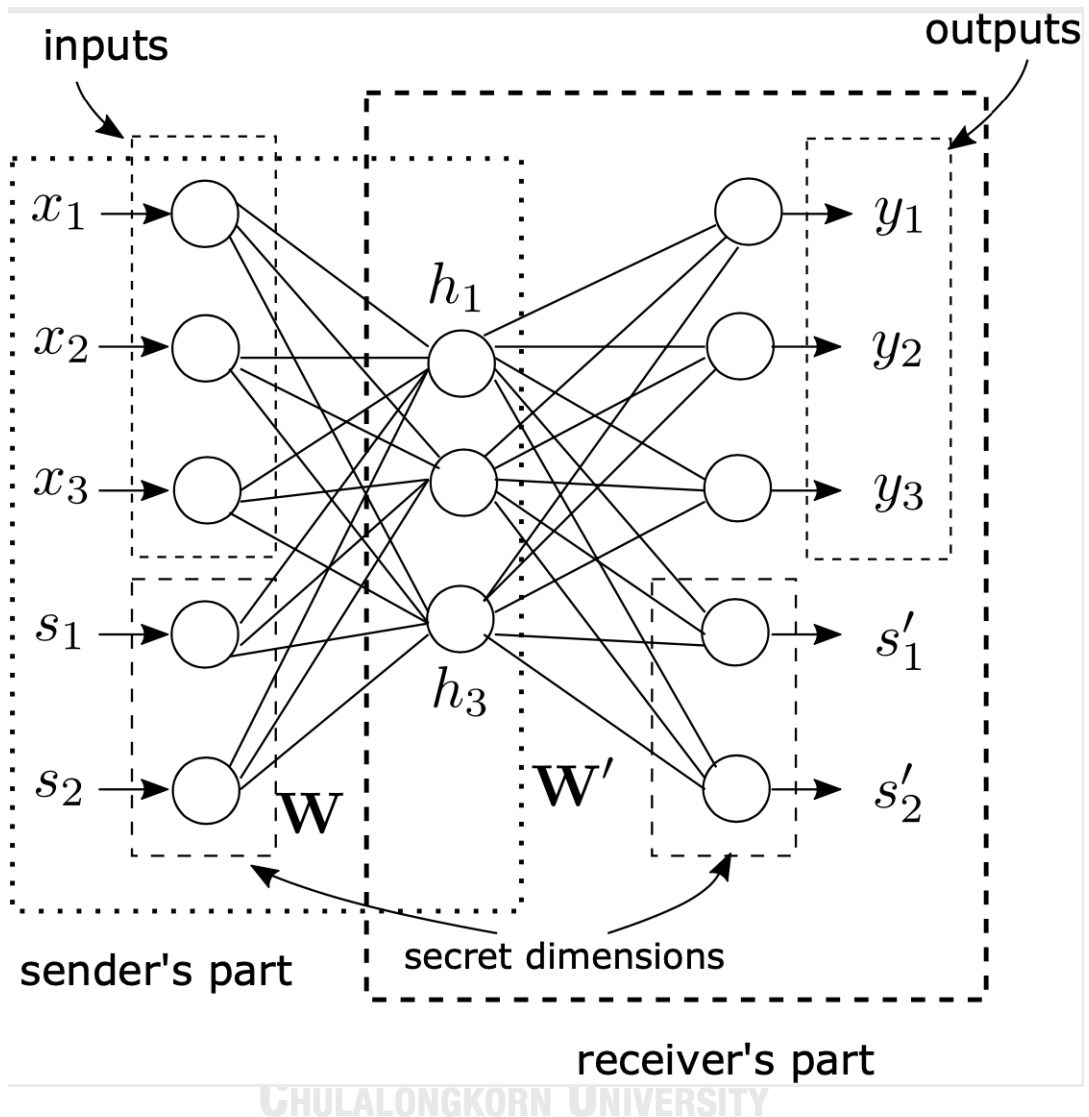


Figure 3.2 The structure of our model.

3.3 Model Design

In the proposed method section, the picture we showed only has one hidden layer. If the input dimensions is large (more than 1,000), the cryptosystem will use the network with deeper layers. Comparing with other models, our model design is more flexible which could be adjusted according to the specific situation.

Since our cryptosystem is used for textual message, the output needs to be the same with the input. So the input of each character will be one training data, as well as one input node, the label of training data is itself, so the whole process is unsupervised learning [16]. According to this, we designed the cryptosystem mainly depending on DNN (deep neural network). For our future study, if we resized the input data, we could also use other popular neural network structure to realize our cryptosystem [17]. For example, if we change the 1-D input data to 2-D pictures, then we are able to deal with the input data as the images. For images, CNN (convolutional neural network) [18] is a good choose to select the main feature of the input data. Besides this, the reinforcement learning could also help to realize our cryptosystem since the reinforcement study could study according the feedback of our data [19].

3.3.1 Scale the data

Our cryptosystem uses the integer as the input data. While if you want to use the min-max scaling to normalize the data to binary data, it will be faster if the input dimension is large (more than 1,000) to encrypt the information. Most of the other model used the technique of normalization [20] to make the gradient descent converge quicker towards the minima.

While the benefits of using the integers is that we could easily check and compare the decoded text with the plaintext. And it is also more friendly for the sender and receiver to use the integers. Since Unicode is the extent of the ASCII code, for the same alphabet, the ASCII code is the same with its Unicode value. So another advantage of using integers as the input can transfer from the ASCII code to Unicode easily without so much change and vice versa.

For our future study, for image and video input information, we consider taking normalization to change the integer data to binary data to faster the training process.

3.3.2 Running Time

The main time used for the message is to find the suitable structure of the model. In the section 4, we introduced the way to find the suitable number for the secret dimension and the hidden nodes with the minimum loss. This part of the training time is decided by the hyperparameters we set in the model, such as the learning rate, the **EarlyStopping**, the number of epochs. The accurate time is different for all kinds of lengths of the message and hyperparameters. Since our cipher is multiple blocks cipher, so it is free to set the block size. If the user has set the block size, such as two users agreed to encrypt every 10 symbols as the plaintext in this month. Then the structure of the model could be decided in the beginning after found the suitable number of secret key and the number of hidden dimensions. So the first part of the model just needs to be run once in this month to decide the structure of the cryptosystem model at first. Later, the users just need to encrypt and decrypt the message based on the built mode. For the short message, if the input dimension is less than one thousand (1,000), the whole cryptosystem could be trained in less than 5 seconds with GPU. If already get the suitable number of secret dimensions and the number of hidden nodes, the encryption part could be finished in 1 second and the same for the decryption part. For long message, if the plaintext has more than one thousand (1,000) while less than twenty thousand (20,000) symbols, the input could be resized otherwise the loss will high. After the input dimensions have been adjusted, the cryptosystem could use different strategy depends on the hardware to compile and train the model. After that, the encryption time and decryption is similar to the short message.

3.3.3 Message Length change

Different from our model, the encrypted string length changes from N to 2^N after encryption in [5]. The hidden layer of our cryptosystem is plasticity. And we used the block cipher, the block size could be set or unfixed. Each block of the

message after being encrypted, the cipher could be larger, equal or less than the original one.

Normally if the input message is less than one thousand characters, it is recommendable to take the second encryption in the cryptosystem or combine other encryption method as the second encryption to make the whole system harder to be hacked. In more detail, the length of every encrypted block changes from the input layer to the hidden layer, which is the encryption part and turns back to the original size from the hidden layer to the output layer after decryption.

3.3.4 Encryption And Decryption key

As for the encryption key, it is a matrix which composed of random numbers and used between the input layer and the hidden layer in the cryptosystem. Additionally, without second encryption, the key is part of the cryptosystem. Furthermore, the decryption key is the matrix which composed of random numbers as well, and it used between the hidden layer and the output layer. It should be emphasized that when the length of the encryption key is precisely as long as the encrypted message, our model is the one-time pad cipher and in theory unbreakable under such situation. However, in terms of the program, the treacherous built-in random function will generate pseudo-random numbers. These may be clues to a solution for attack the encryption and decryption key for one attacker. It was interesting to note that there is no specific relationship between the cipher and the plaintext. In other words, the cipher can be longer, equal, or shorter in relation to the original message.

3.3.5 Fail Rate Of Decryption

Different from other works in the cryptosystem, our model does not have the fail rate. In [6, 8] they use symmetric encryption systems, and the neural network structure will be changed during the training. At the same time, our model sets the parameters of neural networks in the beginning without the training and

learning process. In [6, 7], the training of the neural network failed half of the time. And in [8], the system needed about 25000 trails to get a robust result, and the fail rate is 1/3. There was no unmatched text in large amounts of experiments with plain text, which proves that our model is robust, and it must get the correct decipher according to the proven mathematics.

3.3.6 The Secret Key

The secret key plays several vital roles in our cryptosystem. On the one hand, we could flexibly adjust the size of input if the dimension of the input is too large. For the secret key, not only it could be the noise adding into the original data, it could also change the input dimensions flexible to meet the requirement of cryptosystem.

Except this, the secret key could be used as the hash function to check if the receiver receives the correct data. If one attacker modified the data on the way, the secret key will also be changed and the receiver will know that the data has been cracked. In other word, it makes the cryptosystem safer and the receiver could check the cipher as well.

For our future study, we will use the cryptosystem to encrypt the image and video data. While if we just add one pixel to one RGB picture, the picture could not be resized to the original width and height. So the secret key could be used in two ways. One is the secret key could be used as the noise data, adding to some pixels of the picture without changing the original height and width. The other is we can add more pixels to change the picture size, and it could be resized to the new height or new width. For instance, we could add one row of the pixels to the picture so after resized, the height of the picture should plus one to get the new image.

Except these, since the secret key could be string or random numbers, it could take a lot of information to the receiver. If we use the timestamp to

generate the random numbers for the secret key, the receiver could get the time information with the sender. The generated random numbers could also add some random information such as the weather today to make the cipher become more complex and hard to crack. Besides, the secret dimensions could also be a good interface for other coded information such as Morse code.

There are also many types of design for the secret key of the cryptosystem. For example, you could use the Huffman code to code the plaintext first, and store the information in the secret key. When the receiver gets the cipher, the decoded plaintext needs to use the information in the secret key to decrypt again.

For our cryptosystem, we just use the normal way to design the model. The secret key of our cryptosystem consists of some random numbers. According to various situation, it is also flexible to adjust and change the number of secret dimensions.

CHAPTER 4 EXPERIMENTS

4.1 Data set

We construct a plain text of various lengths for the data set.

One sample is like figure 4.3, which is only the mix of all common symbols.

The benefits of our cryptosystem is that even just a few data it could work. So if you do not have a dataset, just create some toy examples and used in the cryptosystem directly, adjust it and tuning the hyperparameters of the model.



Figure 4.3 One sample in the samples directory.

Our cryptosystem could be extended to use Unicode easily. So we also prepared the Chinese, Japanese, Korean and Thai language in the data set to test the cryptosystem, as shown in the figure 4.4. In each other language directory, the subdirectory is similar like we prepared in the English directory. From the testing result, our cryptosystem could also work well with the Unicode representation.

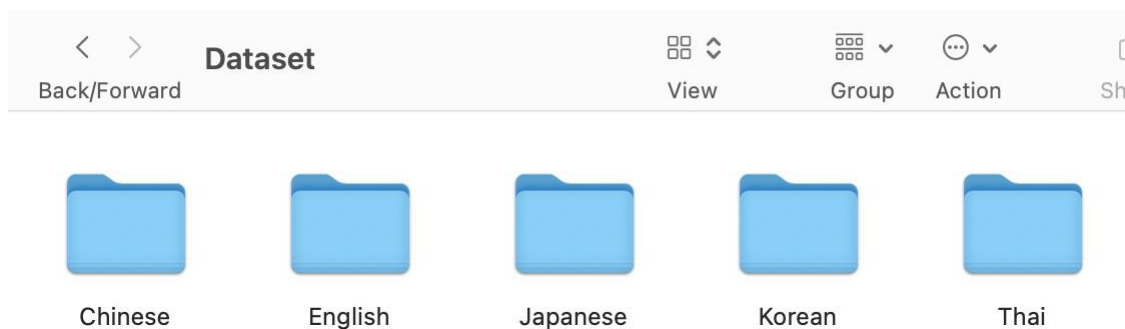


Figure 4.4 Dataset

In the experiment, the following three samples with varying numbers of characters (including symbols such as space, ?, and!) were encoded and decoded to demonstrate how our method works. Encoding and decoding details are provided below.

4.2 Encryption part

Message 1: *Hello, World!* with secret key "13 10 108 111 118 101"

Message 2: *What do you want to eat in the evening?* with secret key "33 116 182 143 154 196 127"

Message 3: *How about we eat in the same restaurant we ate yesterday evening?* with secret key "85 172 180 133"

Leaky-Relu activation function was used for the hidden and output layers. After training the network 1000 epochs for each number of hidden neurons and each number of secret dimensions, the cryptosystem would pick up the suitable number

of hidden neurons and number of secret dimensions automatically according to the loss. It should be noted that if there are many numbers of hidden neurons providing the lowest loss depending on the input message and the number of secret dimensions, we will choose a random one as the number of hidden neurons, which could also be determined by the sender.

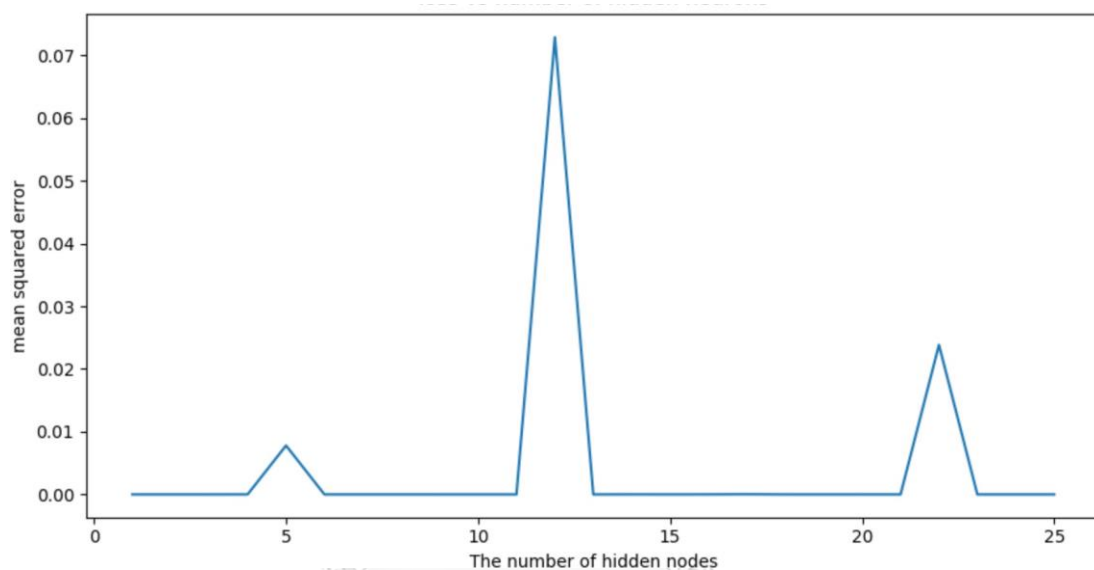


Figure 4.5 Loss vs number of hidden neurons for *Hello, World!*.

After the size of encrypted messages is determined, the size of the secret dimensions must be selected to match the loss.

Figure 4.5 indicates the relation of loss and the number of hidden neurons and secret dimensions for the first message. The subfigure (a) in figure 4.5 shows the mean squared error (loss) changes versus the number of hidden nodes.

The number of hidden nodes can range from one to twice the number of input neurons to test. In this range, for each integer, one model will be compiled and trained on the GPU or TPU, resulting in faster process of encryption.

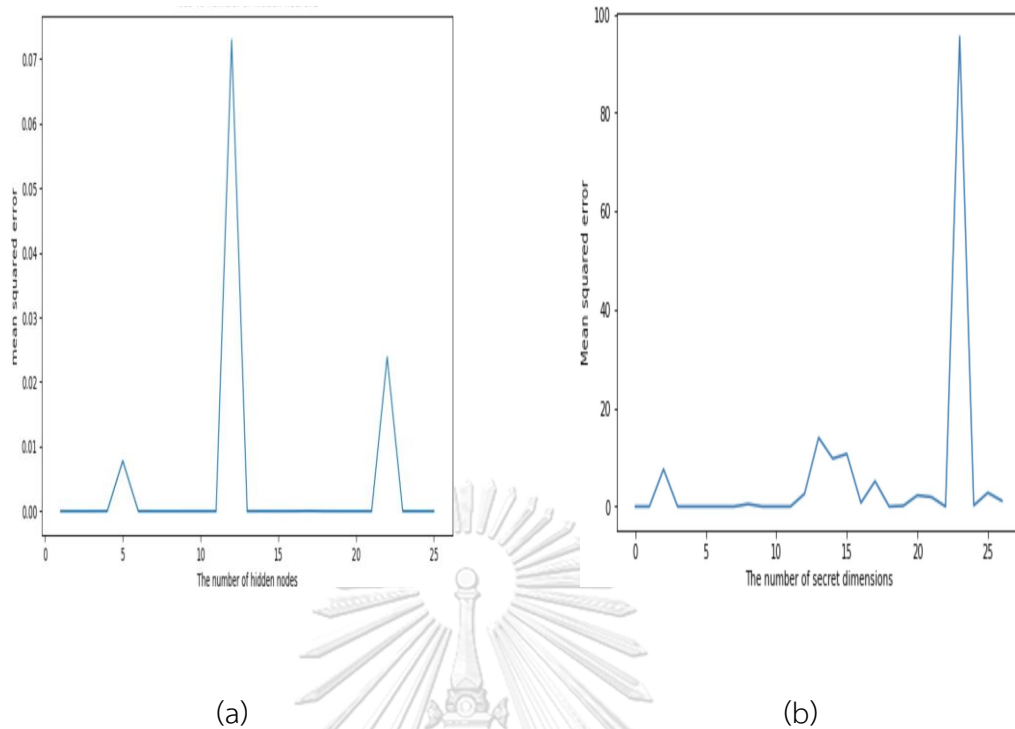


Figure 4.6: Loss vs number of hidden neurons and secret key for message one

Every loss in the subgraph (a) was less than 0.5. Adam's initial learning rate was set at 0.01, and the number of secret dimensions was determined as 6 including "13 10 108 111 118 101". Figure 4.6 summarizes the minimum mean squared error (loss) with the change of the number of hidden neurons. The graphs also have comparable shapes for the same message testing.

When the number of secret dimensions was less than three or greater than eight, the loss was unstable for subgraph (b). When other parameters were unchanged, we got the minimal loss with six secret nodes. From the experiment, the minimal loss might occur in the region between 1 and number of input neurons.

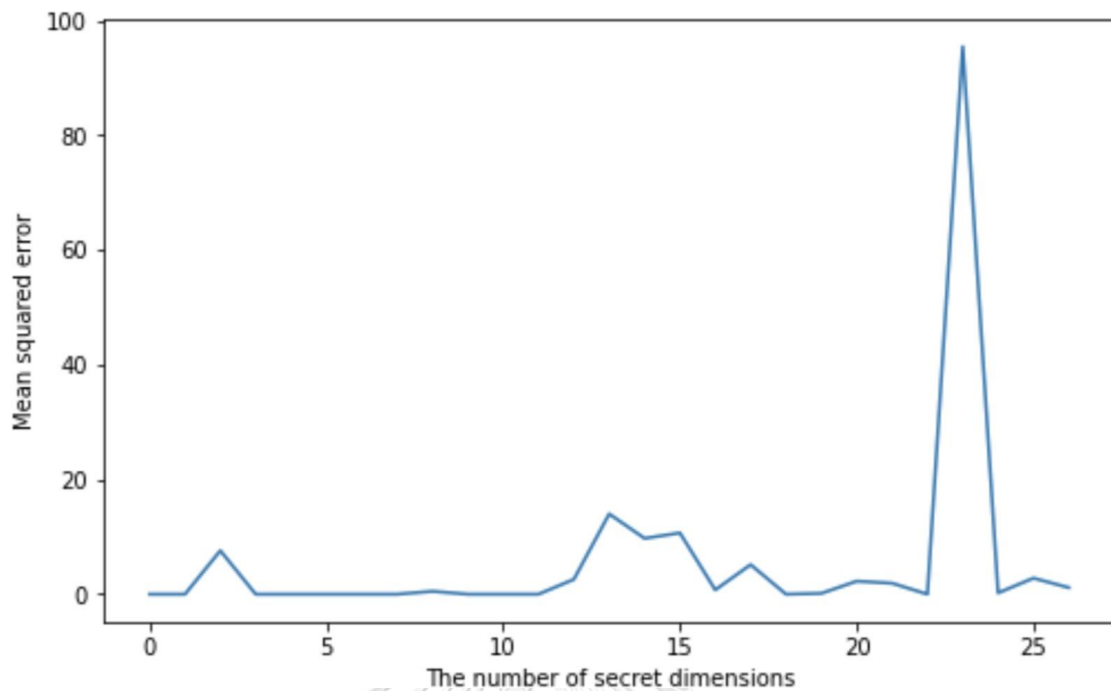


Figure 4.7 Loss vs secret key for *Hello, World!*.

If the number of hidden neurons is fixed, then the size of the secret dimensions must be determined in relation to the loss. Figure 4.7 illustrates the relationship between loss and the number of secret dimensions. Because it required most of the time and effort to train the proper amount of secret dimensions and hidden nodes for one model, the model after trained for the first time could be used as a pre-trained model which could be reused. Transfer learning is the particular field under consideration here. And when a pre-trained model is used in the cryptosystem to generate a new message, all that required is to alter the last layer or add multiple additional levels. To accommodate the new input message with the same length, the model just needs little modification. However, pre-trained models are not frequently employed in the industry or academic community today.

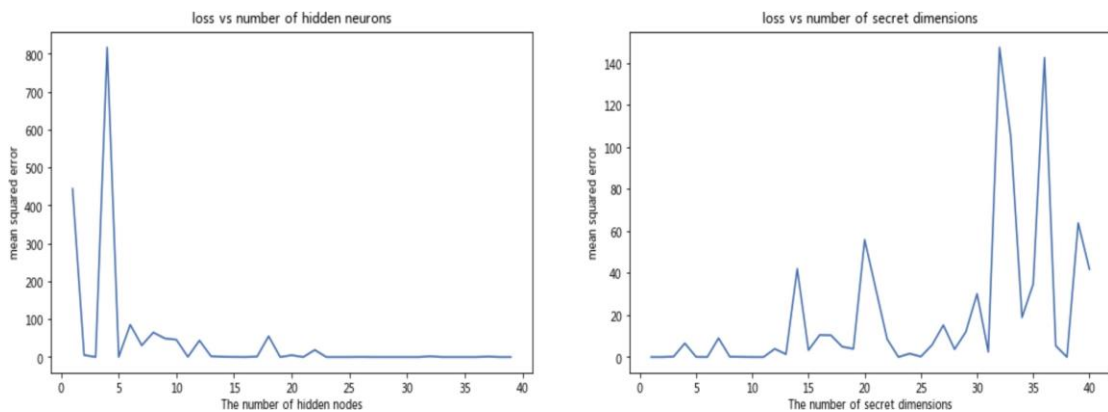


Figure 4.8 Loss vs hidden neurons and secret key for What do you want to eat in the evening?.

A representation of the loss of the second message as compared to the number of hidden neurons and secret dimensions is shown in Figure 4.8. When the hidden nodes exceeded the number of nodes above 19, the loss became stable. After that, the cryptosystem picked up the number of hidden nodes as 20, and trained the neural network again for getting the suitable number of secret dimensions. When the number of secret dimensions (secret key) reached 6, the loss was minimal. When the number of secret dimensions was more than 11, the instability increased. The secret dimensions and the number of hidden neurons for the second sample were set as follows: "33 116 182 143 154 196 127". And the loss of the second message was 1.1216 in one experiment.

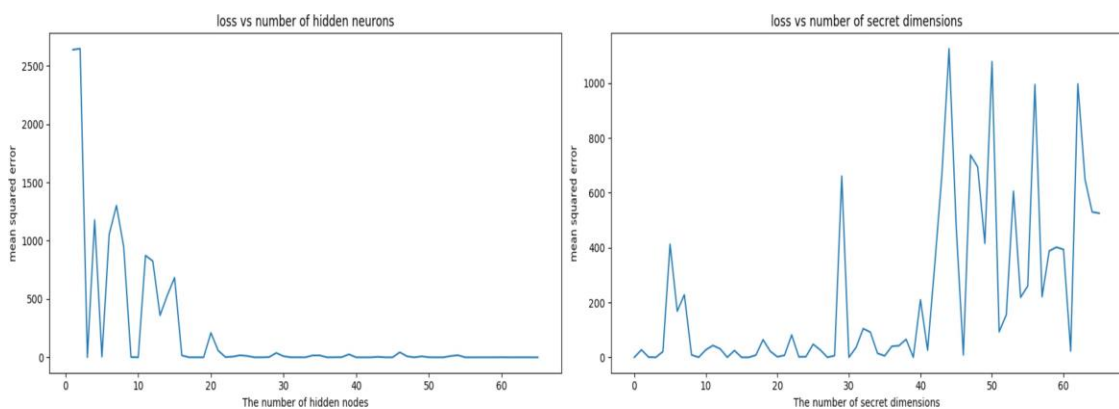


Figure 4.9 Loss, hidden neurons and secret key for How about we eat in the same restaurant we ate yesterday evening?.

The third message's loss versus hidden neurons and secret dimensions are displayed in Figure 4.9. We obtained the least loss when the number of hidden neurons was 62. After setting the number of hidden neurons to 62, we were able to obtain the number of secret dimensions as 4 from the experiment according to the loss. The value of the secret key can be established arbitrarily or according to predefined rules agreed upon by the sender and receiver. It might be a single string or a collection of digits. The activation function for the hidden and output layers was implemented using the Leaky-Relu function, with the alpha parameter initialized to 0.2. For the third message, the secret dimensions were set to "85 172 180 133" and the number of hidden neurons was set to 62.

The encryption and decryption time for each sampled message are summarized in Table 4.1.

Based on numerous trials, we recommend to test number of hidden neurons between half to whole of the number of input neurons to obtain lowest loss. For the range of secret key, it should be appropriate to test between one and half of the neural network's input neurons.

Table 4.1 Encryption and decryption time of three examples

Samples	# Input neurons	# Secret keys	# Hidden neurons	# output neurons	Encryption time (sec)
message 1	13	6	10	1 9	2.294
message 2	39	6	20	4 5	2.186
message 3	65	4	62	6 9	2.697

4.3 Decryption part

The receiver uses the upper part of the corresponding model to decrypt the cipher. For the first message, the output **O** was "Hello, World! 13 10 108 111 118

101". The string part Y is "Hello, World!" and the secret dimensions S' is "13 10 108 111 118 101". For the second message, the decrypted text Y is "What do you want to eat in the evening?" and the secret dimensions S' is "33 116 182 143 154 196 127". For the third message, the decrypted text Y is "How about we eat in the same restaurant we ate yesterday evening?" and the secret dimensions S' is "85 172 180 133". All output messages and secret keys are correctly decrypted. Figure 4.10 summarizes the number of input nodes, number of secret nodes, and number of hidden nodes with the loss of each message. The loss shown in the Figure was rounded to 3 decimal places.

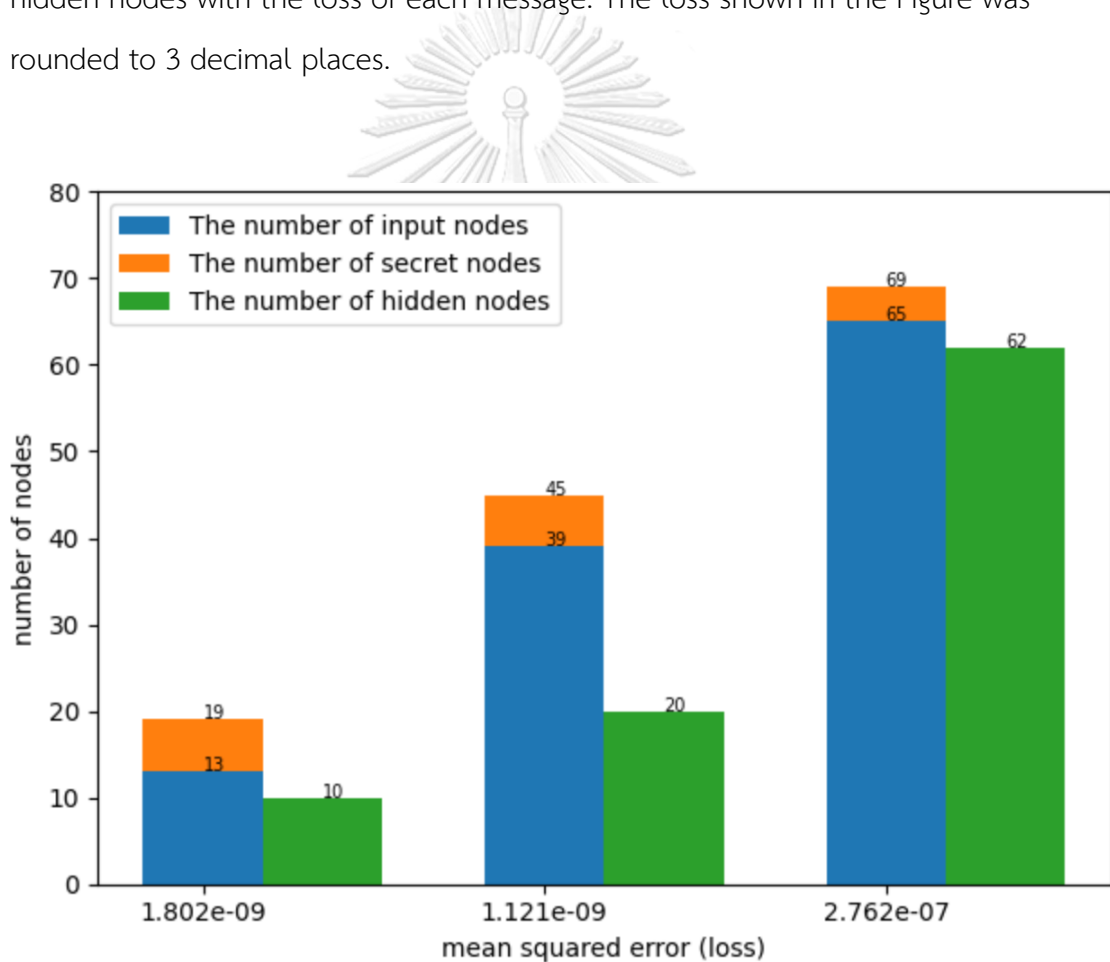


Figure 4.10 Network structure to encrypt and decrypt three example messages.

4.4 Testing environment

The whole cryptosystem uses the TensorFlow 2.4.1 to build with the main coding language is python 3. We build several models both in the local and on the

cloud environment. In the local, the main computer our team used is one MacBook Pro (15-inch, 2018), whose system is macOS Big Sur with the processor 2.6 GHz 6-Core Intel Core i7, and the memory is 32 GB 2400 MHz DDR4. There is one built-in GPU in the computer: Intel UHD Graphics 630 1536 MB. If the cryptosystem used for the plaintext less than 1000 characters, the use of GPU in the experiment will not make such a big difference. If we need to train thousands of characters together, the speed of CPU is far slower than using GPU in the experiments.

The cloud environment such the Google Colab was not used in our experiment due to the service charge. There are also some limits for the normal users to use GPU and TPU, such as the Jupyter Notebook will lose the connection if the page did not use for too long time. And the maximum using time for GPU and TPU is around 12 hours, although we can deploy the cryptosystem to several google users to make the cryptosystem work all the time with GPU or TPU. For message not longer than 1000 characters, (one character is one input node to the cryptosystem), the time of the whole process including the encryption and decryption is not so much different with CPU, GPU or TPU. While for long message which is more than 1000 characters, normally using TPU is faster than just using the GPU, and GPU is faster than just using the CPU.

4.5 Training strategies

Normally, when the input data is huge (more than 10,000 input dimensions trained once for one block) and full utilization of the computer resources is needed, it would be beneficial to employ some strategies as the followings.

4.5.1 Weights and biases initialization

The first step to build the model is initializing the weights and biases. The weights and biases should all be initialized as random numbers whose range is the range of the input data which will be dealt with later in the cryptosystem. For

our model, one cannot use the zero initialization since it will make all the outputs become zeros.

If the test data is only English alphabet with symbols, the range of the weights and biases can be 0 to 1. Since when the weights and biases is larger than one, the cipher or final output should still be dealt with, which should be in the range of ASCII code. If one uses the Unicode as input data, the weights and biases could be 0 to 1 as well (cannot be initialized as 0 to make it converge to 1). There are also some other advanced techniques that one can use if the initialization code by oneself, not by the application program. For example, **He initialization** [21] is a good choice for ReLu activation function used in the cryptosystem, which could help to train the model faster with large input data and avoid slow convergence. In our cryptosystem, the initialization code already has been optimized with the Google's TensorFlow API.

4.5.2 Input size

When the input message is more than 10,000 characters, if we take all one character as one input node, and if one input nodes will be dealt with as one dimension, so the input will be a one by 10000 matrix, and it will take longer time to train and hard to get every decoded result correct after training. Since it is fast for the mode to train the short messages, if we can change the input to one 100 by 100 matrix, which means we resize the input dimension to 100, while deal with 100 groups of such data together, the time for the cryptosystem is very short.

From the experiment, the cryptosystem can deal with the maximum input dimension is around twenty thousand (20,000). While for the input message, the cryptosystem is able to deal with much large data since the input could be resized. For instance, there is one input message which has one hundred thousand (100,000) symbols, the input matrix could be resized to 1,000 by 100, which means we change the input dimension to be 100, and the cryptosystem will deal with 1,000

groups of such data together. Such way is proved to be highly efficient in the experiment. It will also be used in the image and video cryptography for future study.

4.5.3 Learning rate

The initial learning rate was set to be 0.001 in the beginning. For different input dimensions, the learning rate, number of epochs will be adjusted. Different from the constant learning rate, there are two ways to adjust the learning rate: schedule learning rate and adaptive learning rate. For the first one, it normally includes time-based decay, exponential decay and step decay.

For our model, since we use the Adam, we applied the adaptive learning rate method. The range of the learning rate in our model is from $1e-8$ to 0.9, and it is adjusted by hand with experiences. If we would like to freeze the hidden layer weights during training, the optimizer will use the stochastic gradient descent (SGD) algorithm with different learning rate schedules and initial momentum as 0.9 to train the model. However, the default optimization algorithm is Adam.

4.5.4 Loss function

For binary values, which means if we used the normalization to the input data, the loss function could choose the sum of Bernoulli's cross-entropy method instead of Adam optimization algorithm. For binary data, the structure of the model also needs to adjust, especially the activation function in the model. Instead of Relu or Leaky-Relu activation function, the softmax function is better for the cryptosystem if the input dimension is large, while the Relu activation could be trained faster to get the result if the input dimension is not so large.

In the experiments, we mainly used the method that used integers as the input data, which uses the mean squared error loss function is more suitable. And for real values, the activation function of the output layer can not use softmax

activation function which would only produce the value between 0 and 1. And when the output is larger (i.e. larger than 100), the output will be all one, and the same for the small number, the output will all be zero.

4.5.5 Optimization function

If one wants to freeze the layer weights and biases (the encryption key) during training, the optimization function could choose the Stochastic Gradient Descent (SGD) algorithm. Otherwise, it is suggested to use the other more popular gradient descent optimization algorithm: Adaptive Moment Estimation (Adam) in the cryptosystem. In our model, we just used the Adam.

There are some other optimizers that improved from SGD algorithm which can also used in our cryptosystem: Momentum, Adagrad optimizer and RMSprop with Adagrad etc. Momentum helps SGD to solve the local optimum problem that dampens oscillations as well as accelerates SGD in the connected direction.

Adagrad is well-suited for dealing with short message because of the updates of the weights bias will also change according to the learning rate.

RMSprop with Adagrad divides the learning rate by an exponentially decaying, so it only relies on the average of current and previous gradient, normally, the learning rate of RMSprop with Adagrad is smaller than the Adagrad algorithm mentioned above. Whatever optimization function you pick up, paralleling, batch normalization and distributing strategy could make it more powerful and the specific optimizer depends on the model and condition.

4.5.6 Early stopping

The EarlyStopping of TensorFlow could set the monitor and patience to stop the model earlier if the model did not meet the requirements. For example, if we set the monitor is the loss of the model, and the patience is equal to five. Then

when we train the model, if the loss of the model did not go down for 5 epochs, the training would be stopped.

The *min_delta* parameter of the EarlyStopping was set to be **0.0001** in the beginning, which means minimum change in the monitored quantity to qualify as an improvement in the training process.

4.5.7 Callback

The callback function used one or more functions as the parameters, and it will be dealt as the argument passed to another function. And in Machine Learning, the callback is the function that could be applied at various stages of the training procedure.

Except Google's callback interface, we can also implement our own callback functions. Our own build callback function could be inherited from the original callback so that our own implementation of callback could use all the functions that could be used in the built-in callback interface.

And if we add the test data, the confusion matrix could be calculated. And based on the confusion matrix, we could log it as the data summary for per-epoch callback. If that, we will also add the per-epoch callback to the callback function.

In the next subsection, we will introduce the TensorBoard, including the "ModelCheckpoint" in TensorFlow which is able to periodically save your model during the training process and the EarlyStopping we mentioned in above subsection, they are all being used as the parameters for the callback tool in our cryptosystem.

4.5.8 TensorBoard

TensorBoard is one of the most powerful and useful visualization tools. In our cryptosystem, we use it as the tool to track and visualize the mean squared

loss. Except this, the TensorBoard could be used to check the plaintext, cipher and decoded test easily. Besides, if we want to check the weights and biases of the hidden layer, it is convenient to check the change of the histograms of them over time. The usage of TensorBoard in our cryptosystem is like the following.

At first, you need to use the command *%load_ext tensorboard* to load the TensorBoard notebook.

Secondly, clear out data in the previous logging directory and set the position to put the logging data. Next, you can define the basic TensorBoard callback which will put in the callback (reference last subsection) parameter. In the end, you are able to use the command *%tensorboard -logdir* adding your logging directory path to start the TensorBoard. After open the TensorBoard, there are several common modes you can pick up depends on your purpose that we mentioned above: CUSTOM SCALARS, HISTOGRAMS, MESH, TEXT, PR CURVES and PROFILE. The TensorBoard also supports to use SSH tunneling for working on a remote server.

In TensorBoard, you could check the scalars (such as *epoch_loss*) of our cryptosystem. The tooltip sorting method could be adjusted to default, descending, ascending and nearest. And for horizontal axis, it is free to set the STEP, RELATIVE or WALL for it. In the model's scalar graphs on TensorBoard the "Smoothing" parameter is the exponential moving average of all the real scalar values in our model. In the settings of TensorBoard, you could set reload period or pagination limit for reloading data. The logging data can also add other useful information such as the specific time to complete the model.

And one of our models' TensorBoard is like the figure **4.11**. In the figure, we could find that the mean squared loss is large in the beginning, which is more than 10,000 for this model. And the mean squared loss decreased very fast in one epoch. From the picture, we could find that each epoch also stores the information about the date and time to finish it.

The Scalars tab shows changes in the loss and metrics over the epochs, which could be used to track other scalar values such as training speed and learning rate.

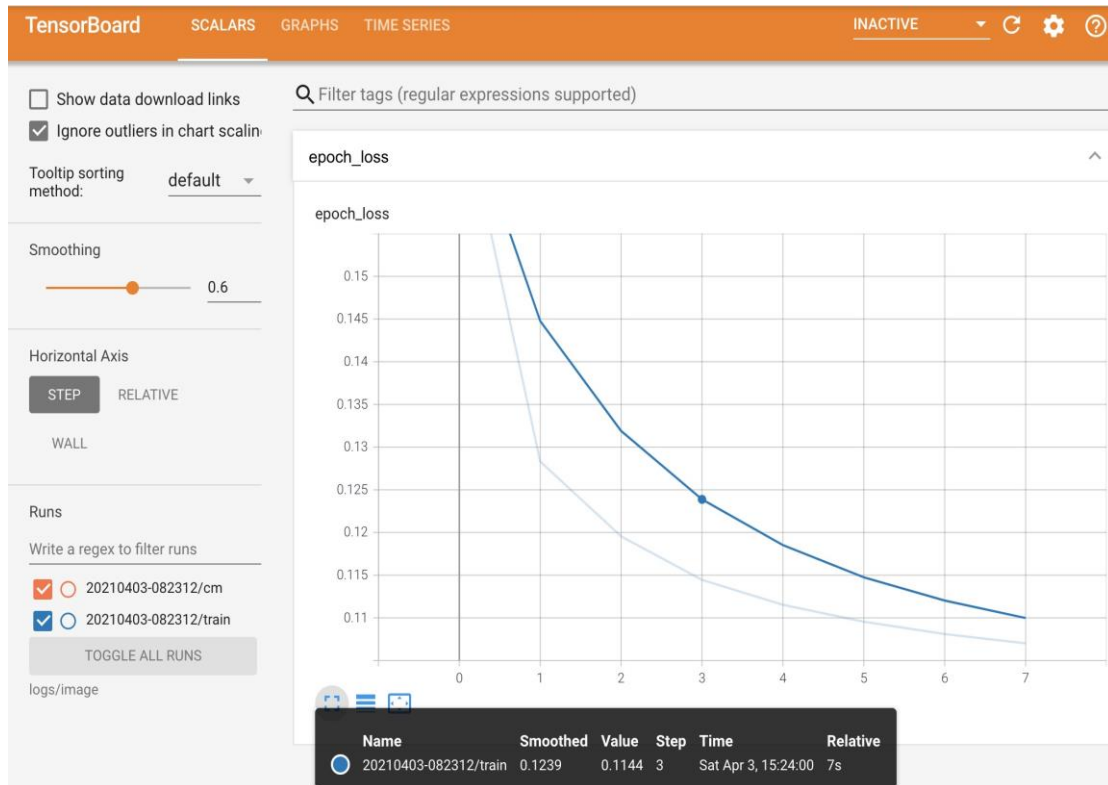


Figure 4.11 TensorBoard of one model of our cryptosystem.

The "graphs" tab on TensorBoard is used to show the model layers. One can use the "graphs" tab to check if the architecture of the model looks as intended, as shown in figure 4.11. As shown in figure 4.12, we can trace the inputs and separate the structure graphs according to their structure, different devices, computing time, used memory, TPU compatibility and XLA (Accelerated Linear Algebra) Cluster which is a domain specific compiler that could accelerate cryptosystem with potentially no source code changes. XLA Cluster is a simple way to start using XLA in the compiled and executed process of the cryptosystem model, which just needs to add several lines of the code.

While in our normal cryptosystem in the lab, we mainly use the common mode with GPU to accelerate the models.

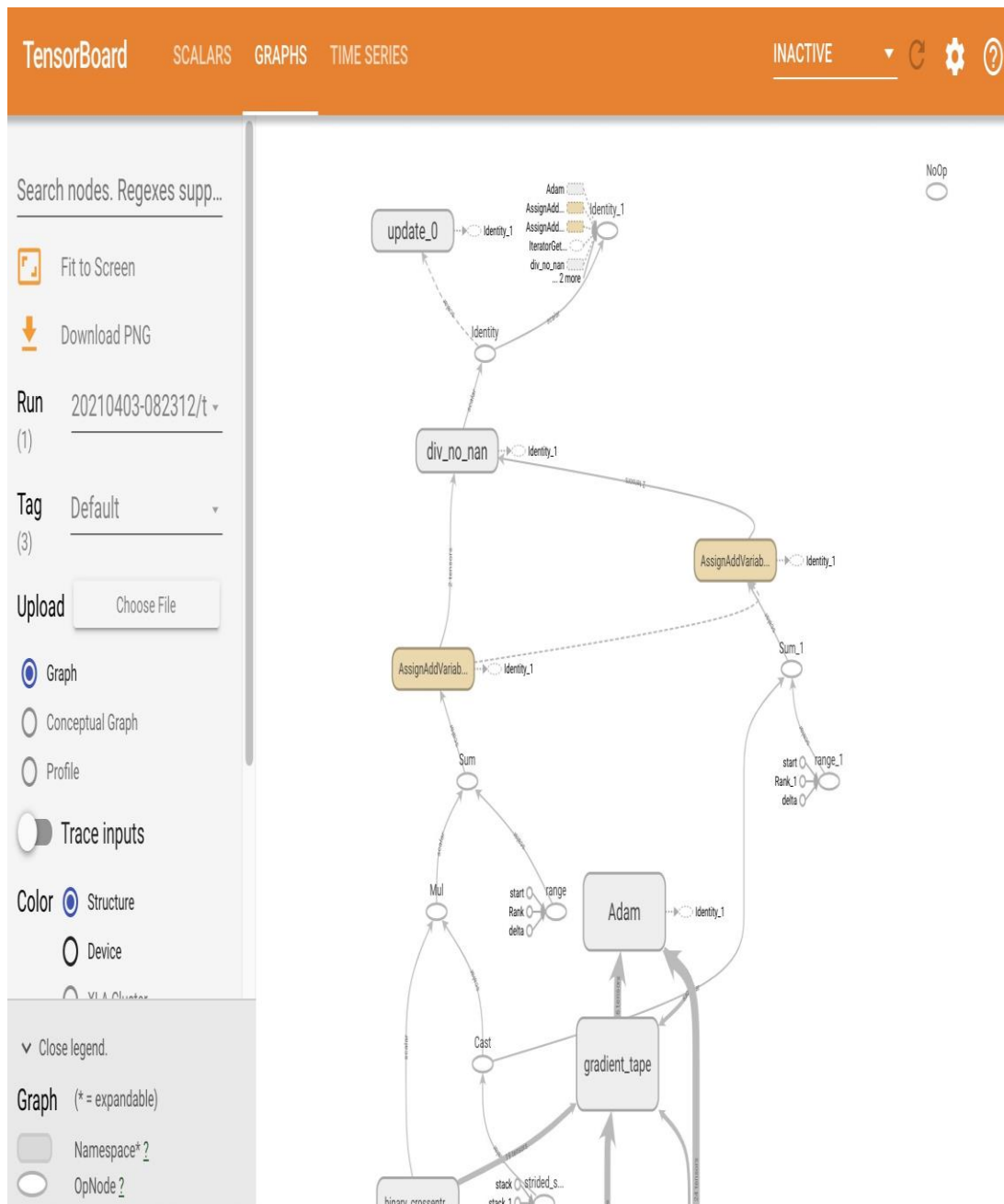


Figure 4.12 The GRAPHS tab of TensorBoard.

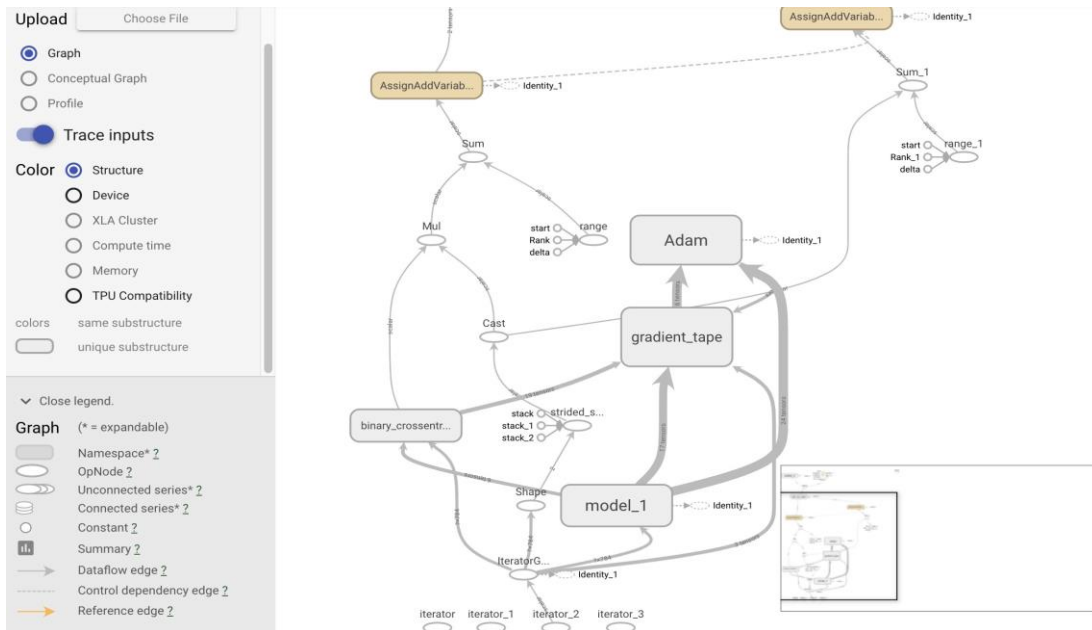


Figure 4.13 The GRAPH of architecture of the model on TensorBoard.

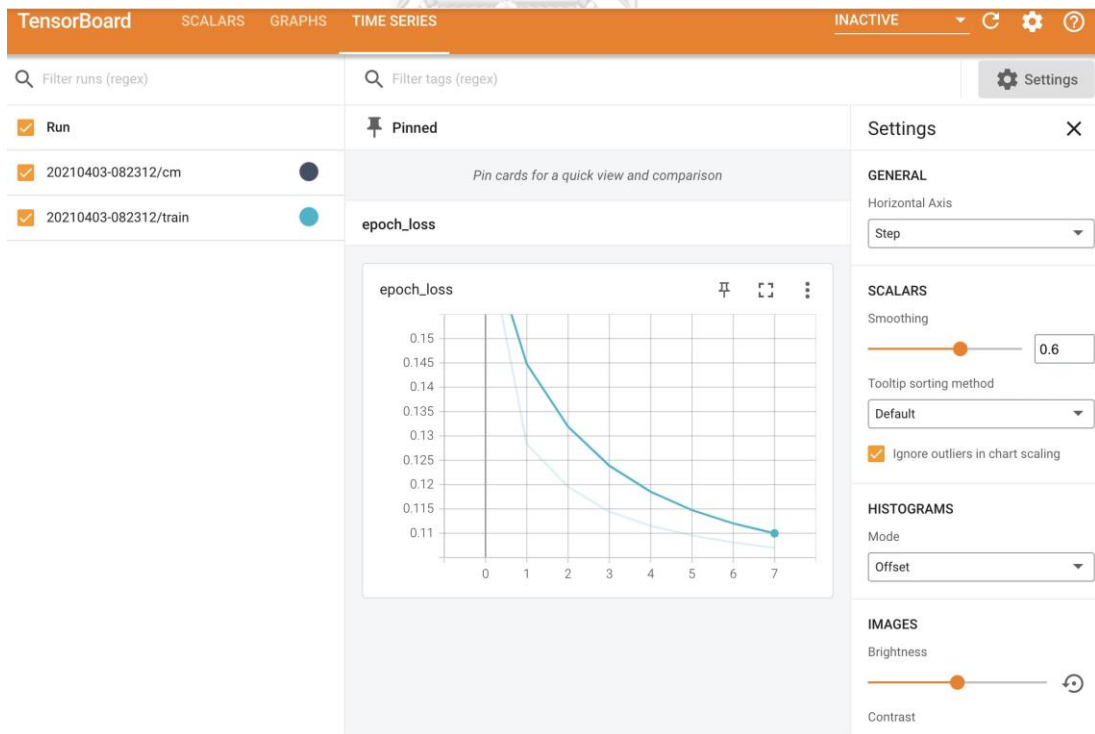


Figure 4.14 Time Series tab on TensorBoard.

The architecture of one model could be show like figure 4.13. About the Time Series tab, the data of one experiment as shown in figure 4.14. The number of epochs is 7. And with the time passing, the loss became smaller. There are some settings just beside the *epoch_loss* image which could be adjusted.

In conclusion, TensorBoard is one of the most powerful tools that used in the training. And we introduced some common setting for the TensorBoard which is only part of its great features. We suggest you to go to the official website of TensorBoard to look up the latest information and have a better command more skills. While using TensorBoard is not all rosy, there are also some limitations for using it.

1. difficult for one team to use when needs to set the environment;
2. If you want to track various experiments, cannot perform data and model visualization;
3. cannot visualize and log some other data formats, such as or normal HTML, video or audio;
4. there is no workspace and user management.

4.5.9 Device strategy

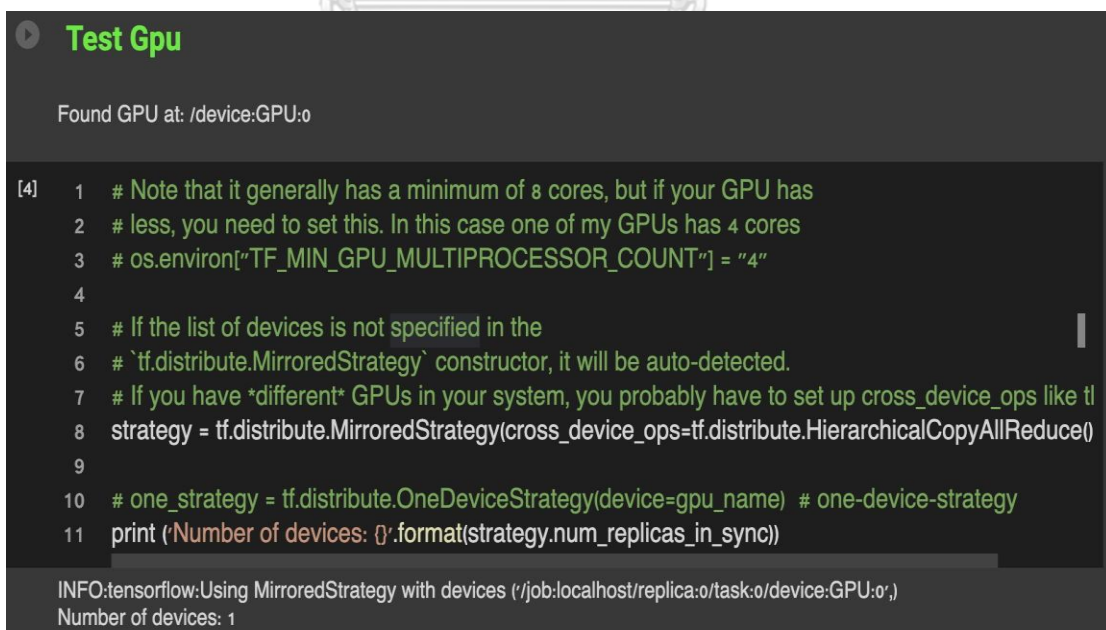
When the input data is huge, there are several ways to choose to faster the training process depends on the hardware. If the device only have one GPU (Graphics processing unit), the cryptosystem will use one device strategy of TensorFlow for large input data. Normally, the GPU has a minimum of eight cores, while if the GPU has less than 8 cores, it could also be set in the beginning when the cryptosystem use this one device strategy.

As shown in figure 4.15, if the system has more than one GPU, the cryptosystem could set up cross device strategy with the "MirroredStrategy" of

TensorFlow which is able to make full use of multi GPU. The other synchronous distributed strategy similar to the "MirroredStrategy" called "MultiWorkerMirroredStrategy" of TensorFlow could implement training across multiple computers, each has potentially multiple GPUs. Another strategy called "ParameterServerStrategy" of TensorFlow is a common data-parallel training method to scale up the model, which used on multiple machines as well.

If the computer only has one GPU, except using the "MirroredStrategy" (can be used on one or more than one GPU), "CentralStorageStrategy" is another choice while is not flexible as "MirroredStrategy". "CentralStorageStrategy" will place all operations and variables on the single GPU, and it also trains the model synchronously. Compared with "MirroredStrategy", the "CentralStorageStrategy", the variables are not mirrored, instead they are stored on CPU and operations are replicated across all local GPUs.

In a word, there are many strategies offered by Google TensorFlow that you can choose according to your working situation.



```

▶ Test Gpu
Found GPU at: /device:GPU:0

[4] 1 # Note that it generally has a minimum of 8 cores, but if your GPU has
    2 # less, you need to set this. In this case one of my GPUs has 4 cores
    3 # os.environ["TF_MIN_GPU_MULTIPROCESSOR_COUNT"] = "4"
    4
    5 # If the list of devices is not specified in the
    6 # `tf.distribute.MirroredStrategy` constructor, it will be auto-detected.
    7 # If you have *different* GPUs in your system, you probably have to set up cross_device_ops like tl
    8 strategy = tf.distribute.MirroredStrategy(cross_device_ops=tf.distribute.HierarchicalCopyAllReduce())
    9
   10 # one_strategy = tf.distribute.OneDeviceStrategy(device=gpu_name) # one-device-strategy
   11 print('Number of devices: {}'.format(strategy.num_replicas_in_sync))

INFO:tensorflow:Using MirroredStrategy with devices (/job:localhost/replica:0/task:0/device:GPU:0;)
Number of devices: 1

```

Figure 4.15 GPU MirroredStrategy: Supports Synchronous Distributed Training With Multiple GPUs On Single Machine.

Meanwhile, the TPU (Tensor Processing Unit) is similar, one can also set the clusters for training the model with distributed "TPUStrategy" of TensorFlow. Since until 2021, April, the TPU is still only available on Google's Colab, we could only test the TPU on the cloud. The TPU strategy is shown in figure 4.16. There will be some warning if the TPU has already been initialized, since it may cause previously created variables on TPU to be lost.

```

If want to use TPU, set up TPUs and initialize TPU Strategy

1 # Detect TPU
2 try:
3     tpu_address = 'grpc://' + os.environ['COLAB_TPU_ADDR']
4     tpu = tf.distribute.cluster_resolver.TPUClusterResolver(tpu_address) # TPU detection
5     tf.config.experimental_connect_to_cluster(tpu)
6     tf.tpu.experimental.initialize_tpu_system(tpu)
7     strategy = tf.distribute.TPUStrategy(tpu)
8     # Going back and forth between TPU and host is expensive.
9     # Better to run 128 batches on the TPU before reporting back.
10    print('Running on TPU ', tpu.cluster_spec().as_dict()['worker'])
11    print("Number of accelerators: ", strategy.num_replicas_in_sync)
12 except ValueError:
13    print('TPU failed to initialize.')

WARNING:tensorflow:TPU system grpc://10.32.170.242:8470 has already been initialized. Reinitializing the TPU can cause previously created variables on TPU to be lost.
WARNING:tensorflow:TPU system grpc://10.32.170.242:8470 has already been initialized. Reinitializing the TPU can cause previously created variables on TPU to be lost.
INFO:tensorflow:Initializing the TPU system: grpc://10.32.170.242:8470
INFO:tensorflow:Initializing the TPU system: grpc://10.32.170.242:8470
INFO:tensorflow:Clearing out eager caches
INFO:tensorflow:Clearing out eager caches
INFO:tensorflow:Finished initializing TPU system.
INFO:tensorflow:Finished initializing TPU system.
INFO:tensorflow:Found TPU system:
INFO:tensorflow:Found TPU system:
INFO:tensorflow:*** Num TPU Cores: 8
INFO:tensorflow:*** Num TPU Cores: 8

```

Figure 4.16 TPU Strategy Our team used in the lab

The information of the output is like figure 4.17. From the figure 4.17, we can get that the TPU has 8 accelerators. Similar like CPU, there is one common distribution strategy could train on TPUs and TPU Pods synchronous: "TPUStrategy".

With this strategy, the program cannot use the pure eager execution. So if using the "TPUStrategy" of TensorFlow, several parts of code need to be changed from "GPUStrategy". Since TPU could only use on Google Colab, so we mainly use the GPU strategy in the program which could easily deploy on different platforms to test. With different strategies, it helps us to test and run the program with the scalability of distributed computing.

```

INFO:tensorflow:*** Num TPU Workers: 1
INFO:tensorflow:*** Num TPU Workers: 1
INFO:tensorflow:*** Num TPU Cores Per Worker: 8
INFO:tensorflow:*** Num TPU Cores Per Worker: 8
INFO:tensorflow:*** Available Device: _DeviceAttributes(/job:localhost/replica:0/task:0/device:CPU:0, CPU, 0, 0)
INFO:tensorflow:*** Available Device: _DeviceAttributes(/job:localhost/replica:0/task:0/device:CPU:0, CPU, 0, 0)
INFO:tensorflow:*** Available Device: _DeviceAttributes(/job:worker/replica:0/task:0/device:CPU:0, CPU, 0, 0)
INFO:tensorflow:*** Available Device: _DeviceAttributes(/job:worker/replica:0/task:0/device:CPU:0, CPU, 0, 0)
INFO:tensorflow:*** Available Device: _DeviceAttributes(/job:worker/replica:0/task:0/device:TPU:0, TPU, 0, 0)
INFO:tensorflow:*** Available Device: _DeviceAttributes(/job:worker/replica:0/task:0/device:TPU:0, TPU, 0, 0)
INFO:tensorflow:*** Available Device: _DeviceAttributes(/job:worker/replica:0/task:0/device:TPU:1, TPU, 0, 0)
INFO:tensorflow:*** Available Device: _DeviceAttributes(/job:worker/replica:0/task:0/device:TPU:1, TPU, 0, 0)
INFO:tensorflow:*** Available Device: _DeviceAttributes(/job:worker/replica:0/task:0/device:TPU:2, TPU, 0, 0)
INFO:tensorflow:*** Available Device: _DeviceAttributes(/job:worker/replica:0/task:0/device:TPU:2, TPU, 0, 0)
INFO:tensorflow:*** Available Device: _DeviceAttributes(/job:worker/replica:0/task:0/device:TPU:3, TPU, 0, 0)
INFO:tensorflow:*** Available Device: _DeviceAttributes(/job:worker/replica:0/task:0/device:TPU:3, TPU, 0, 0)
INFO:tensorflow:*** Available Device: _DeviceAttributes(/job:worker/replica:0/task:0/device:TPU:4, TPU, 0, 0)
INFO:tensorflow:*** Available Device: _DeviceAttributes(/job:worker/replica:0/task:0/device:TPU:4, TPU, 0, 0)
INFO:tensorflow:*** Available Device: _DeviceAttributes(/job:worker/replica:0/task:0/device:TPU:5, TPU, 0, 0)
INFO:tensorflow:*** Available Device: _DeviceAttributes(/job:worker/replica:0/task:0/device:TPU:5, TPU, 0, 0)
INFO:tensorflow:*** Available Device: _DeviceAttributes(/job:worker/replica:0/task:0/device:TPU:6, TPU, 0, 0)
INFO:tensorflow:*** Available Device: _DeviceAttributes(/job:worker/replica:0/task:0/device:TPU:6, TPU, 0, 0)
INFO:tensorflow:*** Available Device: _DeviceAttributes(/job:worker/replica:0/task:0/device:TPU:7, TPU, 0, 0)
INFO:tensorflow:*** Available Device: _DeviceAttributes(/job:worker/replica:0/task:0/device:TPU:7, TPU, 0, 0)
INFO:tensorflow:*** Available Device: _DeviceAttributes(/job:worker/replica:0/task:0/device:TPU_SYSTEM:0, TPU_SYSTEM, 0, 0)
INFO:tensorflow:*** Available Device: _DeviceAttributes(/job:worker/replica:0/task:0/device:TPU_SYSTEM:0, TPU_SYSTEM, 0, 0)
INFO:tensorflow:*** Available Device: _DeviceAttributes(/job:worker/replica:0/task:0/device:XLA_CPU:0, XLA_CPU, 0, 0)
INFO:tensorflow:*** Available Device: _DeviceAttributes(/job:worker/replica:0/task:0/device:XLA_CPU:0, XLA_CPU, 0, 0)
Running on TPU [10.32.170.242:8470]
Number of accelerators: 8

```

Figure 4.17 Output of the TPU after using the TPU strategy in the lab.

The advantage for us to use the cloud is that it is convenient and economic for our team to deploy and train many models with various hyperparameters on multi devices. The disadvantage is that there will be some delay if we compile and train the model. And the process will be interrupted if the session is inactive for too long or the computer lost the internet.

For our cryptosystem, when we encrypt the message and start to compile and train the model, the data sent to the service center of the cloud needs at least one second, and the vice versa. After the model finished the training, the Jupyter notebook update the new data needs more than one second as well, which depends on the internet speed. So even for a very short message, the Jupyter notebook on the Colab will take more than 3 seconds to get the cipher. While for long message (more than 1000 input dimensions) to be encrypted, the time delay could be omitted if the GPU/TPU of the cloud is powerful enough.

4.6 Why choosing TensorFlow

There are many open source models used in the industry now. We picked up TensorFlow to deploy the cryptosystem mainly based on the following reasons:

1. Our cloud environment mainly uses the Google's Colab which support the built-in

TensorFlow and help our team to connect with other products of Google such as Google Drive to help to test the model.

2. TensorFlow 2.x is one of the most popular framework for deep learning used, so it will be easier to help other people use our code, and we can learn from other excellent programmers as well.

3. Google's Colab already installed the latest TensorFlow, and it supports free GPU and TPU testing for as long as 12 hours now, the Google's Colab Pro has been available in the following countries until 2021 April: Brazil, United States, India, Canada, Germany, France, Japan, United Kingdom, and Thailand.

4. There are many great APIs in the TensorBoard including the above strategies that we introduced, such as weights and biases initialization, already been optimized in the TensorFlow.

4.7 Save and load models

If the block size is fixed or the sender and the receiver will not change the block size during a short time, then for the model structure unchanged, the best model will be saved after the training process of the first time. The model saved for the fixed block size can also be shared to someone who needs to use the cryptosystem for a short time.

After the new training of the model, the parameters, trained weights and biases of the model all be adjusted again to keep other information secret. If the block size has been changed, then the model will be generated again, the old model cannot be used to hack the new information.

About saving the model, the TensorFlow's built-in function uses the checkpoint callback option, you could choose save the weights only or save the model. The frequency of saving the model could be picked, such as save the model once every 10 epochs or just save the best model after the training. The entire model could be saved in several formats: HDF5 format and JSON format. HDF5 is the basic standard save format and JSON format is the other common format used in TensorFlow. The saved model could be load whenever want to be use. For the cryptosystem, some unused model could be exploited as the tool to attack the cryptosystem in the lab as well.

4.8 Maintenance cryptosystem

If the cryptosystem did not work in the training process sometimes, you can refer the section 4.5 and try some new training strategies to improve the model. While after training, there should be some predictive program to do the automated anomaly detection for the cryptosystem. In other words, the maintenance program is part of the cryptosystem as well. Normally the model should work well, and if the output is different with the input, there will be some warning or error. The

cryptosystem should check by itself to maintenance. And if the program did not find any problem, you should check it to understand which part is wrong, wrote the maintenance documents for other programmers to check. There should be error history, cryptosystem operating situations, the data of the model and repair history in the maintenance documents. The information about the error should be accurate and specific. In addition, the error should be fixed as soon as possible. If the cryptosystem did not work in the training process sometimes, you can refer the section 4.5 and try some new training strategies to improve the model. While after training, there should be some predictive program to do the automated anomaly detection for the cryptosystem. In other words, the maintenance program is part of the cryptosystem as well. Normally the model should work well, and if the output is different with the input, there will be some warning or error. The cryptosystem should check by itself to maintenance. And if the program did not find any problem, you should check it to understand which part is wrong, wrote the maintenance documents for other programmers to check. There should be error history, cryptosystem operating situations, the data of the model and repair history in the maintenance documents. The information about the error should be accurate and specific. Furthermore, the error should be fixed as soon as possible.

CHAPTER 5

ANALYSIS OF OUR MODEL

5.1 Complexity analysis

To improve the security and robustness of cipher, the suggested network structure has the following 6 configurable components. These components serve as mutually recognized keys.

1. **Secret key.** Assume that each number is in radix r and that there are B dimensions. As a result, the set of feasible hidden dimensions is r^B .
2. **Number of hidden layers.** Number of hidden layer L could be defined freely as another key sent to the receiver. As a result, the number of layers could be in the range $[3, n]$. Thus, for total L hidden layers, the total number of hidden neurons is $\sum_{i=1}^L |H_i|$.
3. **Number of input message to be encrypted.** The suggested network topology is capable of encrypting any statement regardless of its length in terms of characters. Let l represent the length of cipher, the number of sentences that c characters formed is equal to l^c (independent of their semantics).
4. **Set of weights.** Assume there are L hidden layers and each with $|H_i|$ neurons. As a result, the number of weights is $\sum_{i=1}^{L-1} |H_i| \times |H_{i+1}|$.
5. **Cipher.** Only the first layer's output values are used as the cipher in our experiment. The length of the cipher is $|H_1|$.

Prior to delivering the cipher, the receiver receives the secret key and the information about upper part of the neural network for decrypting the message. There are $r^B + L + l^c + \sum_{i=1}^L |H_i|$ bits in the keys, which are equal to $\log_2(r^B + L + l^c + \sum_{i=1}^L |H_i|)$. There are $\log_2(|H_1| + \sum_{i=1}^{L-1} |H_i| \times |H_{i+1}|)$ bits of the weights and cipher. If the length of the cipher is equal to the length of the plain text (and thus also to the decoded text), and the cipher is composed of true random numbers (adding the true random secret key), the cipher would meet the criteria of the one-time pad (introduced in the section 2 background part), which is impossible to hack.

5.2 Cryptanalysis

Cryptanalysis was used to assess the resilience of the proposed method against any attack. There are three normal ways to crack one cryptosystem: Normal attack, cipher text-only attack and chosen plain text attack. In our experiment, we tried all kinds of ways to attack our cryptosystem while all failed. The following is the detail of the experiment.

5.2.1 Normal attack

We have tried all kinds of software programs or apps on the market to attack our cryptosystem while none of them got a good result in the threshold testing time. First, the normal attack tool failed to attack our cipher since the cipher is all float numbers. Frequency analysis also fails since one alphabet is replaced by a random matrix composed of many symbols without a fixed length.

Suppose one attacker C who grasps all the particular information exchanged between A and B and comprehends the detail about the cryptosystem. In principle, C could start from all the possible permutations. The following is the analysis that breaking the construction of our model is an underlying untoward dilemma.

When C tries to hack the cipher, it is possible to cut the cipher from four bits (M is greater than 3) to $L \times M$ bits, according to the possibility analysis, C may hack what is M while he still needs enormously time to get the numbers of $Z1$. From $Z1$ to X , it is impossible to use the brute force because there are infinite possibilities of the $W2$ (part of the decryption key). In conclusion, the brute force, dictionary attacks and frequency analysis can not work on our method.

5.2.2 Cipher text only attack

A successful hack is observed together with the secret keys and cipher as well as how long it took for the assault to succeed. If the time to decrypt the

cipher is substantially longer than the maximum specified limit or the retrieved key-value pairs are erroneous, it shows that the attacker has failed to break cipher. Though an attacker is aware that the cryptography employs a neural network structure, the intruder has to estimate various factors, including the network topology, all synaptic weights, number of characters, and size of secret key. Thus, in order to imitate the assault process, the following stages are set up.

1. Generate number of hidden and output neurons, as well as synaptic weights randomly

2. Send one cipher to the network and estimate the result.

3. Check the message's accuracy in comparison to the original.

4. **While** there is a mistake **or** time < threshold time **do**

5. Renew the network as well as its weights.

6. Check for accuracy in relation to the original message.

7. **EndWhile.**

In this attacking simulation, the same cipher (in Section Four's Encryption section) were employed.

Message 1: *Hello, World!*

Message 2: *What do you want to eat in the evening?*

Message 3: *How about we eat in the same restaurant we ate yesterday evening?*

To break the encrypted communication, a 2-layered network is employed. The first layer inserts every encrypted message needs to be encrypted

and the last issues the decrypted message. The attack will guess the plain text from the encryption during the attacking procedure.

Table 5.1 Three examples of network structure.

Sentences	Message + Secret keys	Input Nodes	Secret Nodes	Hidden Nodes	Output Nodes	Total Time
Sentence 1	Hello, World! + 13 10 108 111 118 101	13	6	10	19	2.294 sec.
Sentence 2	What do you wantto eat in the evening?33 116 182 143 154 196 127	39	6	20	45	2.186 sec.
Sentence 3	How about we eat in the same restaurantwe ate yesterday evening? + 85 172 180 133	65	4	62	69	2.697 sec.

Table 5.1 illustrates the network structure of the three sentences. To preserve consistency, we utilized the same messages with the same secret key as before to demonstrate the cracking process. The message's secret key is made up of random numbers. If the secret key is an integer, it is simpler to convert it to a string using ASCII or Unicode. And if the model uses data normalization, which converts integers to binary numbers, the output will be binary numbers before being converted back to integers.

Table 5.2 Attack result.

Message 1 (13 characters) with 6 nodes of the secret key		
#guessed neurons	cracking time (secs)	#incorrect characters
2-20	> 30,000	19

Message 2 (39 characters) with 6 nodes of the secret key		
#guessed neurons	cracking time (secs)	#incorrect characters
2-46	> 90,000	45

Message 3 (65 characters) with 4 nodes of the secret key		
#guessed neurons	cracking time (secs)	#incorrect characters
2-70	> 150,000	69

The cracking time, the amount of wrongly guessed characters, and the secret keys are summarized in Table 5.3 for each number of guessed neurons. We have many python scripts to run parallelly on multiple machines for message 1, as shown in picture 5.1.

The attack script will count and compute the erroneous amount of characters as well as the erroneous number of secret key nodes. For message 1, 19 python files will be processed in parallel. And for message 2, 45 python files will be performed at the same time. For message 3, 69 python files will be performed concurrently on various computers. If we put the attacking software in the cloud, the internet latency was taken into account.

The final result is recorded in a single table, and we put the conclusion together as shown in the table 5.3. For all messages, the attack scripts incorrectly predicted all message nodes.

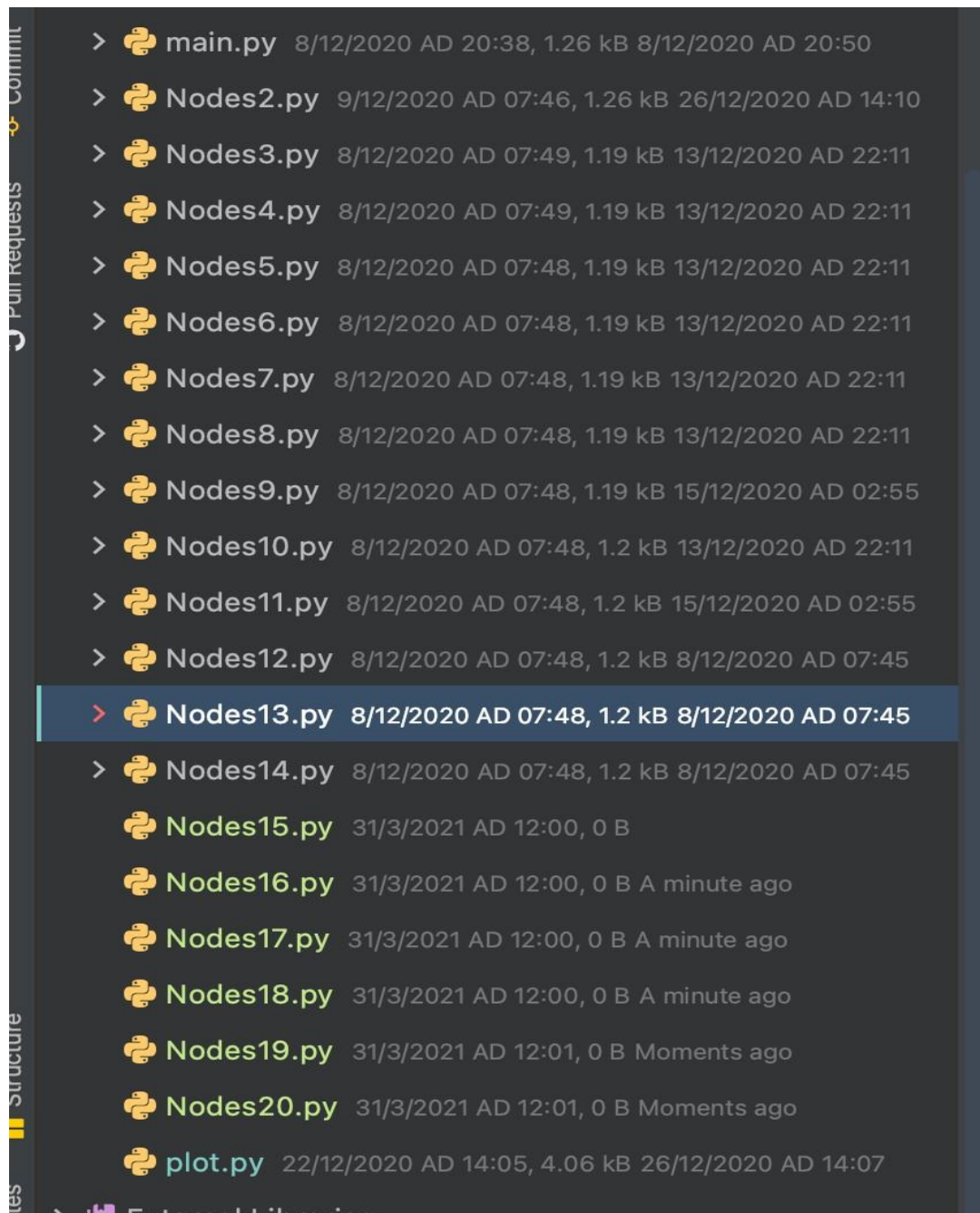


Figure 5.1 The python files we used to test for the message one in local computer.

To save time, we shall run numerous attack scripts concurrently. For example, suppose we guess the nodes from 2 to 13, then execute "Nodes2.py" to "Nodes13.py", as shown in Figure 5.2.

The attack iterates 401,866 times while attempting to determine whether the number of concealed nodes equals 8 in figure 5.2. And the smallest

error remains 19, which is equal to the sum of the number of characters and the number of secret keys. It indicates that all 401,866 assaults failed to get any cipher information. The threshold time for the message one is 30,000 seconds.

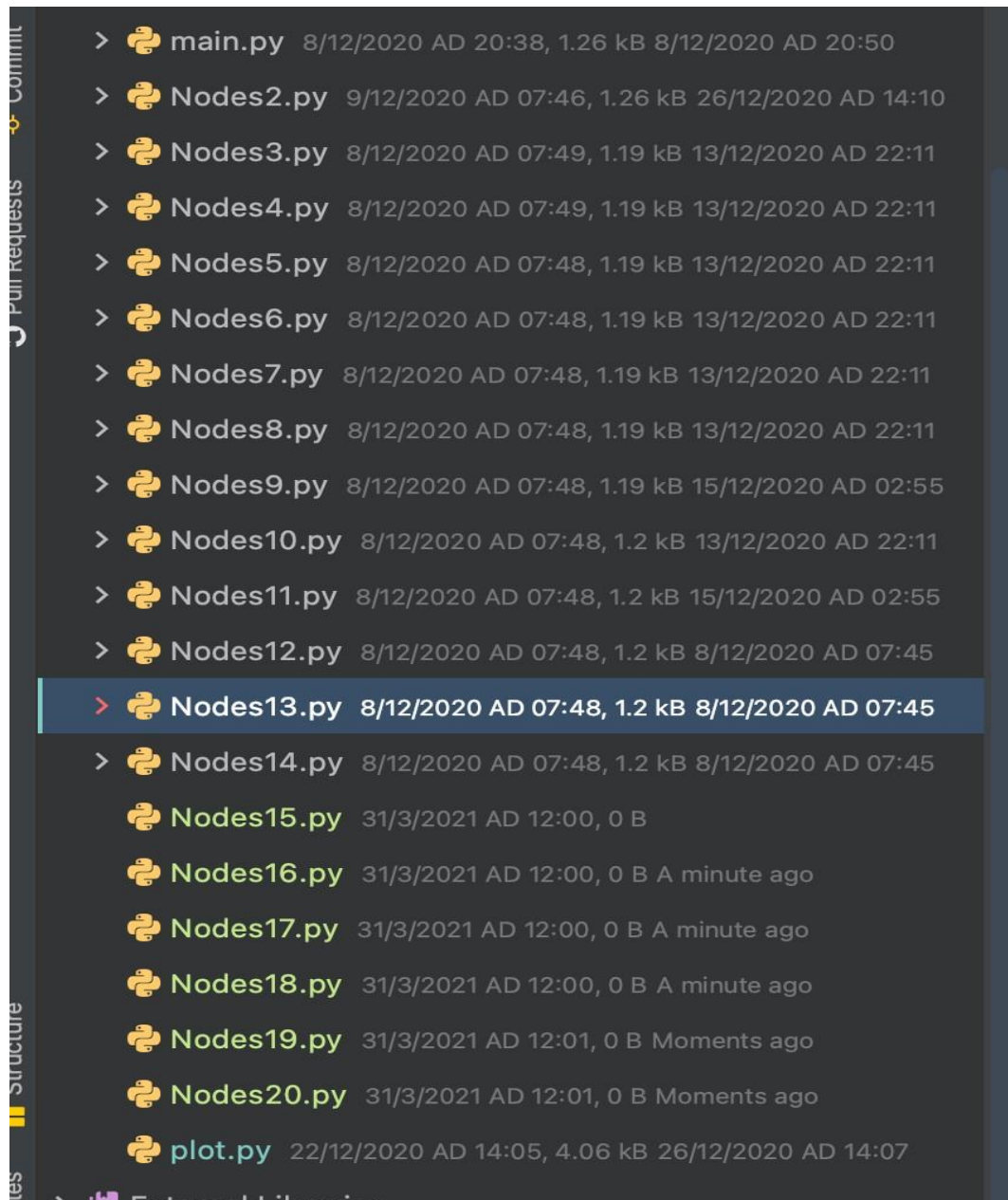
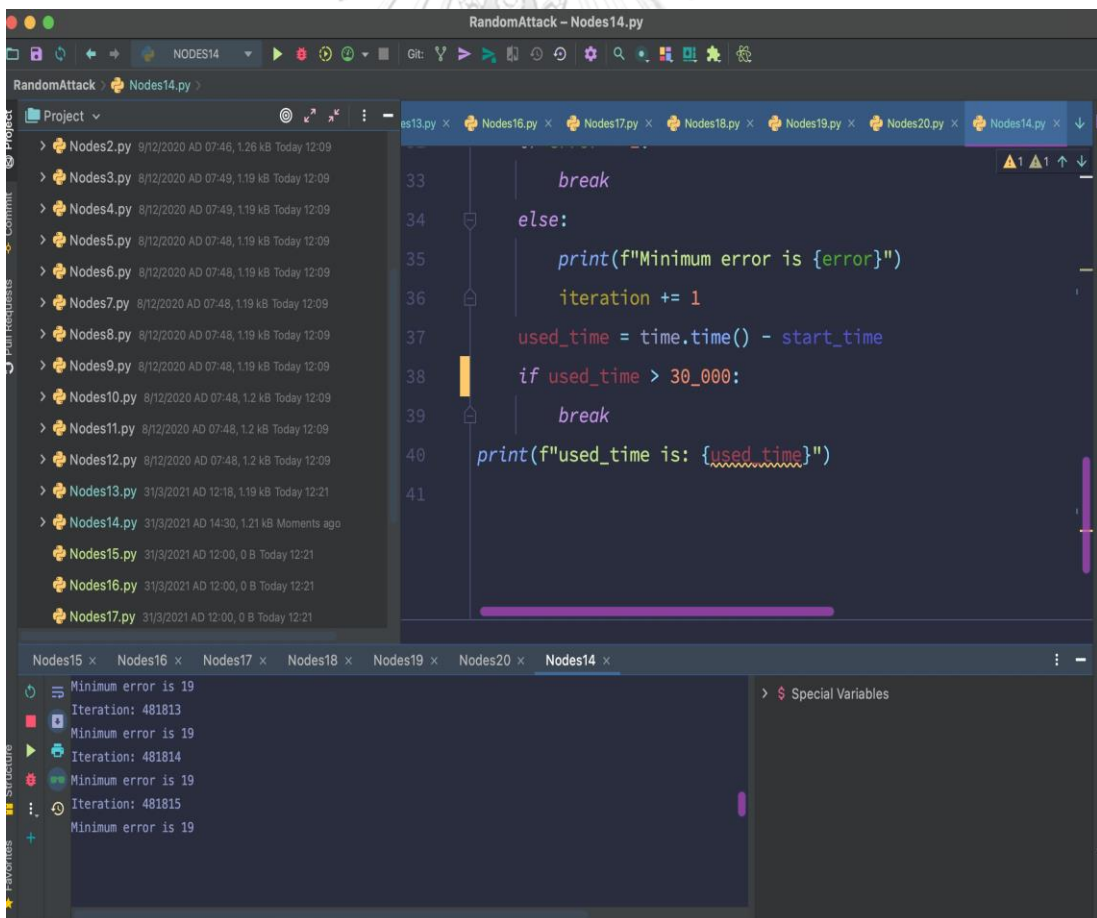


Figure 5.2 Hacking the Message 1 and guessing from nodes 2 to 13.

The "Nodes14.py" to "Nodes20.py" will be performed at a different running time, as illustrated in figure 5.3. To replicate the attack procedure for the brief message, we utilized the local computer as the primary platform. While for

larger messages with more than one hundred characters, we will employ cloud computing capacity such as "Google Colab" with distributed computing, which will allow us to receive the simulation result in less time. In terms of cloud platforms, in addition to "Google Colab," we have utilized IBM's "Watson Studio" ("Jupyter Notebook" comparable to "Google Colab"), "Kaggle Notebook" (a large data research platform), and "Heywhale Notebook" (a platform comparable to Kaggle). Aside from this, each email address may only register for one account on any platform. Each account may create one or more free "Jupyter Notebook" to deploy our application at the same time, allowing our team to test a large message in a short period of time. The simulated attack procedure on cipher text-only assault is similar for other messages in the trials.



```

33     break
34 else:
35     print(f"Minimum error is {error}")
36     iteration += 1
37     used_time = time.time() - start_time
38     if used_time > 30_000:
39         break
40     print(f"used_time is: {used_time}")
41

```

Minimum error is 19
Iteration: 481813
Minimum error is 19
Iteration: 481814
Minimum error is 19
Iteration: 481815
Minimum error is 19

Figure 5.3 Hacking the Message 1 and guessing from nodes 14 to 20.

According to the simulation, all encrypted communications cannot be cracked vs various network structures within the set threshold time (calculated based on message length), which is significantly more than the decryption time.

For the same message, the encryption will be different with a different private key. And, if the sender did not specify the fixed seed for random numbers (weights and biases) in the program, the encrypted message is modified each time the sender uses the cryptosystem to encrypt the same message.

According to our simulated studies, the attacker was unable to get any relevant information within the maximum time limit, implying the proposed method's security and resilience.

5.2.3 Chosen plain text attack

We used the way of [22] to crack the cipher. In the paper, the researchers tried to train one neural network with the cipher text and plain text as input and output. When the loss is less than 0.5, the neural network succeeds in cracking the cipher.

The neural network could be trained with only one block, and the crack time depends on the length of cipher text and plain text. However, the time is considerable to crack two blocks depends on the program used to analyze each block's size. The crack time also varied with different messages.

When we tried several blocks of plain text and cipher text as the input and output of the neural network, the cryptanalysis failed to crack the cryptosystem since the cipher text of each is unfixed. The attacker could not make sure which part of the cipher was for the first block and how large it was for each block.

The range of each block size of the cipher text can range from 4 to a huge number. The neural network would fail to crack the whole cipher text if it could not analyze each block size. Overall, it would be more difficult and

complicated with more blocks of cipher text. The possibilities are already complicated when there are two blocks in the cipher text. And the attacker has to analyze the size of each block of cipher text first to continue the remaining attack.

In a word, breaking the framework of our model can not be finished in linear cryptanalysis time, which proves the security of our algorithm.

In more detail, there are three different situations for the experiment: (1) the number of cipher nodes is less than the number of plain text nodes; (2) the number of cipher nodes is larger than the number of plain text nodes; (3) the number of cipher nodes is equal to the number of plain text nodes. Since the above three examples only show the first situation, we will use another 3 samples to show three different situations that the attacker will meet.

The structure of three examples is shown in table 5.1. When there are multiple blocks in one cipher, it is the same situation and harder to crack. The examples are only part of our crack experiments, while it implies that our cryptosystem is very robust. The chosen plain text attack combined with the neural network is one popular attacking method recently [14] which uses the advantage of strong computation power and learning ability of neural network. Though for a very short message, it is possible that the cipher will be hacked. But in the real life, the cipher will be designed, and normally the cipher length should follow the suggestion of Section 2 (Background), considering Table **2.2** as one reference around 2021 to keep the communication between the sender and the receiver safe.

Table 5.3 Detail of the examples of three situations.

Examples	Input Nodes	Secret Nodes (S)	Message Nodes (X)	Output Nodes	Total Time	Decryption Time
Cipher nodes < plaintext nodes	2	4	10	14	2.89 sec.	1.232 sec.
Cipher nodes > plaintext nodes	20	1	4	5	3.12 sec.	1.445 sec.
Cipher nodes = plaintext nodes	10	1	9	10	1.93 sec.	0.96 sec

5.2.3.1 Number of cipher nodes less than number of plain text nodes

Chosen plain text attack also has a maximum cracking time which set to be the 30,000 (8.34 hours) in the beginning in the attacking program.

For the first example, our cryptosystem encrypted "abandoned\nQcH<" (14 nodes, including X and S, one node is one symbol), got the cipher –314.42844, 70.05509 (2 nodes). The decryption time was 1.232 seconds, and the whole running time was 2.89 seconds with 1000 epochs. The crack program generate the output nodes from 2 to 14 (can also generate more).

When the crack program generated 2 nodes (2 symbols) from the cipher, if no symbol is correct in 8.34 hours (maximum cracking time), then the error will be 14 (the number of symbols in plain text.) Suppose the generated 2 symbols were ac, then the error would be 13. From the result, figure 5.4, we did not get any symbol correct when generating 2 to 14 nodes in 8.34 hours. For different message, the threshold will be adjusted according to their length and hack difficulty. The chosen plain text attack using the method of neural network to attack and suppose the attacker knows the detail of our cryptosystem. While even like this, the

final cracking result is still not good. The attacker did not get any correct node of the message, neither the secret node.

The detail of the cracking process is the attacking program takes one number in one categorial, and use one hot label to classify which number it is. Since our data has no label (or the label is the data itself), so there is no testing data, the one hot coding will be very slow and waste of a lot of space to attack the cryptosystem. The attacking program will change the integers into binary numbers which can accelerate the attacking process, and training the neural network model to crack the cipher. The loss function of the attacking program is binary cross-entropy loss which actually predict the probabilities of each number (one number is one class).

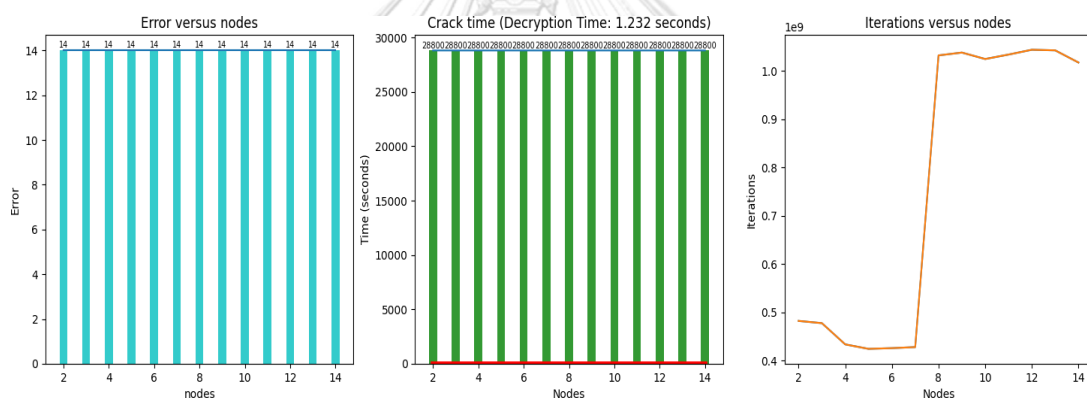


Figure 5.4 2-14 Hack Result.

5.2.3.2 Number of cipher nodes larger than number of plain text nodes

And for second situation, our cryptosystem encrypted "abeZW", got 20 float numbers for the cipher after 1000 epochs in one test. The decryption time was around 1.445 seconds and the whole running time was 3.12 seconds. The crack result is like figure 5.5, we did not get any symbol correct in 8 hours

5.2.3.3 Number of cipher nodes equal to number of plain text nodes

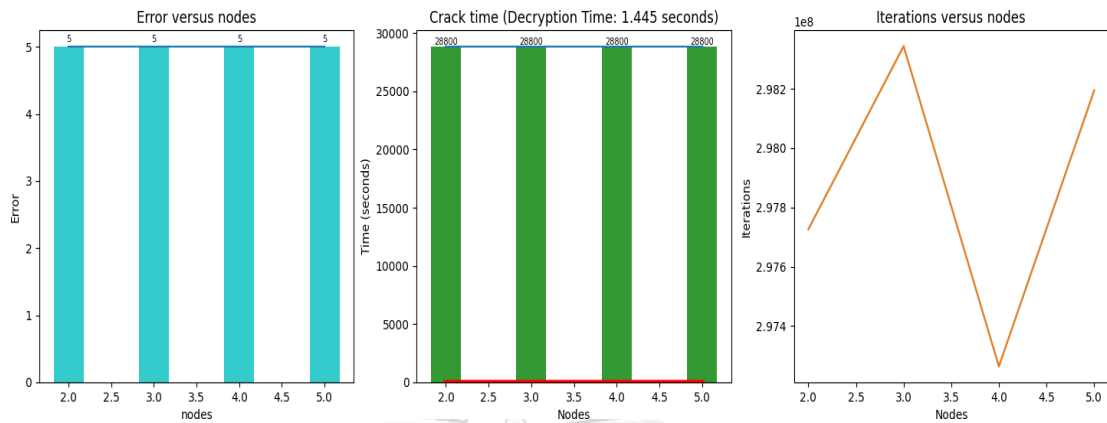


Figure 5.5 20-5 Hack Result.

One node in our example represents one integer (in ASCII code), so even the number of cipher nodes is equal to the number of plain text nodes, does not mean that the length of the cipher is equal to the length of the plain text. When the number of cipher is the same with the number of encrypted message, our cryptosystem encrypted "Thank you!" (10 nodes, including X and S), got 10 float numbers for the cipher after 1000 epochs in one test. The decryption time was around 0.96 seconds and the whole running time was 1.93 seconds. The crack result is like figure **5.6**. Within 8.34 hours, the output nodes did not meet any correct character.

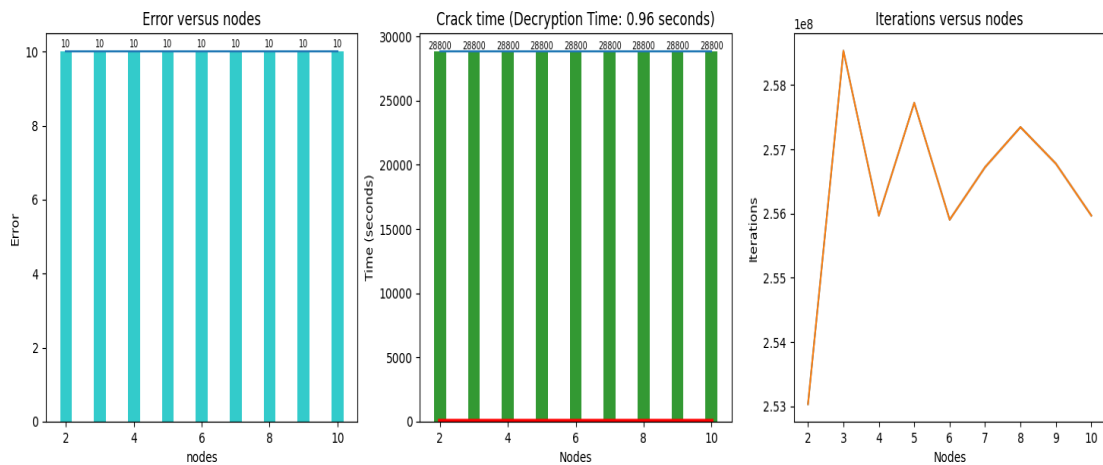


Figure 5.6 10-10 Hack Result.

When there are multiple blocks in one cipher, it is the same situation and harder to crack. The examples are only part of our crack experiments, while it implies that our method is very robust. Compared with other asymmetric cryptography such as DES, AES and RSA, the table 5.4 briefly shows the difference of encryption time, decryption time and cryptanalysis time between our method with other method. Our method used longer time than other methods to encrypt the message while it take shorter time to decrypt the message compare with the RSA which is commonly used in our daily life. RSA, AES and our method are all have high security since these three methods cannot be cracked in a threshold testing time. While our method with the malleable structure is more flexible than RSA and AES.

Table 5.4 Comparative of DES, AES, RSA with our method of encryption time, decryption time and cryptanalysis time.

Algorithm	Size of input (bits)	Encryption Time	Decryption Time	Cryptanalysis Time
56-bit Data Encryption Standard (DES)	1000000	2.8 seconds	1.1 seconds	Under 30 minutes
128-bit-AES	1000000	1.3 seconds	0.8 seconds	More than 3 days cannot crack
128-bit-RSA	1000000	7.2 seconds	4.8 seconds	More than 3 days cannot crack
Our method (flexible for one block)	1000000	12.8 seconds	1.9 seconds	More than 3 days cannot crack

CHAPTER 6

SUGGESTION AND CONCLUSION

6.1 Suggestion of setting cryptosystem

From our experiments, the suggestion for the setting of the cryptosystem is like figure **6.1**.

The first subgraph of Figure **6.1** shows one experiment for one short message (not more than one hundred input dimensions). The loss became small just after several epochs in the training process.

The second subgraph of Figure **6.1** shows the relation between the mean squared error and the number of input dimensions with the number of secret dimensions. When the input neurons (including the input dimensions of the message and the secret dimensions of the secret key) becomes larger, the loss becomes larger as well.

The third subgraph of Figure **6.1** shows the relation between the learning rate and the input neurons (including the input dimensions of the message and the secret dimensions of the secret key). When there is more input nodes, The learning rate should be smaller. In other words, the initial learning rate should set smaller when the number of input dimension is larger.

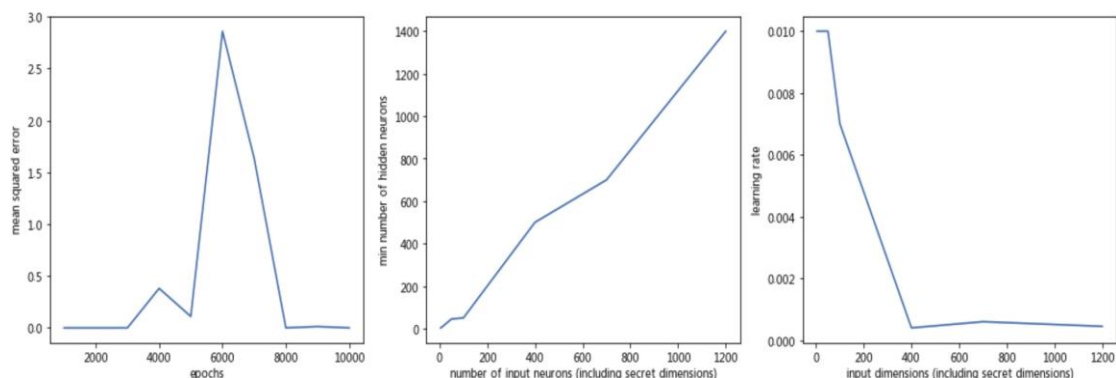


Figure 6.1 Reference when setting the cryptosystem from our experiments.

In more detail, the following number could be taken as a reference. And the data mainly get from the "macOS Big Sur" whose processor is 2.6 GHz 6-Core Intel Core i7 and memory is 32 GB 2400 MHz DDR4. The number maybe different on different testing machine.

When the number of input dimension is around four to ten, the minimal number for hidden dimension is 3 to 7. And the learning rate of Adam or SGD algorithm should be set around 0.01.

When the number of input dimension is around ten to one hundred, the minimal number for hidden dimension should get from 7 to 50. And the initial learning rate of Adam or SGD algorithm should set from 0.007 to 0.01.

When the number of input dimension is around one hundred to four hundred, the minimal number for hidden dimension should get from 50 to 500. And the initial learning rate of Adam or SGD algorithm should set from 0.0004 to 0.007.

When the number of input dimension is around four hundred to seven hundred, the minimal number for hidden dimension could get from 500 to 700. And the suitable learning rate of Adam or SGD algorithm could set from 0.0004 to 0.0006.

When the number of input dimension is around seven hundred to one thousand two hundred, the minimal number for hidden dimension could get from 700 to 1400. And the suitable learning rate of Adam or SGD algorithm could set from 0.00045 to 0.0006.

When the number of input dimension is larger, we did not do the experiment to test while one could take the **6.1** as a reference when the number of input dimension is larger than 1200.

6.2 Suggestion when cryptosystem fails

If it is overfitting when one trains the model, the cryptosystem could still work well if the loss is less than 0.5. And one could set the "patience" parameter in the training process if one uses TensorFlow in the cryptosystem. For example, if one sets the "patience" parameter as 3, then if the loss is less than 0.5 for 3 epochs, the training process will stop, and it could save the time to encrypt for the cryptosystem. When the final mean squared loss (or other loss if one changed the format of the input data) is larger than 0.5, which means some characters could not be reconstructed, then one can tune the learning rate first to try. If the learning rate does not work as well, it is possible that the number of input dimension is too large. If the training process is slow, one can use distributed strategies that we introduced in the Section 4 to take less time for training the model.

6.3 Conclusion

In a nutshell, an innovative asymmetric neural cryptography model dealt with textual messages was introduced in this research to increase the complexity of guessing the cipher. With a malleable neural network topology, our model integrated extra secret dimensions as one key. In terms of time complexity to guess accurate messages, the cryptanalysis test demonstrates the resilience and security of the proposed approach. The future research will concentrate on image and video cryptography.

Appendix I

SAMPLE CHAPTER

A.1 Sample

This is the appendix ...



Appendix II

LIST OF PUBLICATIONS

B.1 International Conference

1. Hangyi Wang, Chidchanok Lursinsap. Neural Crytosystem for Textual Message with Plasticity and Secret Dimensions. In 2021 18th International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology (ECTI-CON), 2021, pp. 27-30, doi: 10.1109/ECTI-CON51831.2021.9454684.



REFERENCES

1. Williams, C.P., *Explorations in quantum computing*. 2010: Springer Science & Business Media.
2. Bos, J.W., et al., *Selecting elliptic curves for cryptography: An efficiency and security analysis*. *Journal of Cryptographic Engineering*, 2016. **6**(4): p. 259-286.
3. Jogdand, R. and S.S. Bisalapur, *Design of an efficient neural key generation*. *International Journal of Artificial Intelligence & Applications (IJAIA)*, 2011. **2**(1): p. 60-69.
4. Ruttor, A., W. Kinzel, and I. Kanter, *Dynamics of neural cryptography*. *Physical Review E*, 2007. **75**(5): p. 056104.
5. Volna, E., et al. *Cryptography Based On Neural Network*. in *ECMS*. 2012.
6. Abadi, M. and D.G. Andersen, *Learning to protect communications with adversarial neural cryptography*. arXiv preprint arXiv:1610.06918, 2016.
7. Zhu, Y., D.V. Vargas, and K. Sakurai. *Neural cryptography based on the topology evolving neural networks*. in *2018 Sixth International Symposium on Computing and Networking Workshops (CANDARW)*. 2018. IEEE.
8. Zhou, L., et al., *Security analysis and new models on the intelligent symmetric key encryption*. *Computers & Security*, 2019. **80**: p. 14-24.
9. Gaffar, A.F.O., A.B.W. Putra, and R. Malani. *The Multi Layer Auto Encoder Neural Network (ML-AENN) for Encryption and Decryption of Text Message*. in *2019 5th International Conference on Science in Information Technology (ICSITech)*. 2019. IEEE.
10. Klimov, A., A. Mityagin, and A. Shamir. *Analysis of neural cryptography*. in *International Conference on the Theory and Application of Cryptology and Information Security*. 2002. Springer.
11. Kelsey, J., et al., *Side channel cryptanalysis of product ciphers*. *Journal of Computer Security*, 2000. **8**(2-3): p. 141-158.
12. Gomez, A.N., et al., *Unsupervised cipher cracking using discrete gans*. arXiv

- preprint arXiv:1801.04883, 2018.
13. Singh, S., *The code book: how to make it, break it, hack it, crack it*. 2003: Ember.
 14. Vincent, P., et al., *Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion*. Journal of machine learning research, 2010. **11**(12).
 15. Xiang, L., et al., *Interpretable complex-valued neural networks for privacy protection*. arXiv preprint arXiv:1901.09546, 2019.
 16. Barker, E., et al., *Recommendation for key management: Part 1: General*. 2006: National Institute of Standards and Technology, Technology Administration.
 17. Wang, L., et al., *ICONIP'02*. Cortex, 2002. **1363**: p. 571.
 18. Dubey, A. and V. Jain, *Applications of Computing, Automation and Wireless Systems in Electrical Engineering*. 2019, Springer Singapore:.
 19. Ramamurthy, R., et al. *Using echo state networks for cryptography*. in *International Conference on Artificial Neural Networks*. 2017. Springer.
 20. Alani, M.M. *Neuro-cryptanalysis of DES*. in *World Congress on Internet Security (WorldCIS-2012)*. 2012. IEEE.
 21. Koturwar, S. and S. Merchant, *Weight initialization of deep neural networks (dnns) using data statistics*. arXiv preprint arXiv:1710.10570, 2017.
 22. Kingma, D.P. and J. Ba, *Adam: A method for stochastic optimization*. arXiv preprint arXiv:1412.6980, 2014.

VITA

NAME Hangyi Wang

DATE OF BIRTH 09 October 1995

PLACE OF BIRTH Zhejiang, ShaoXing City, China

INSTITUTIONS ATTENDED - Department of Mathematics and Computer Science, Faculty of Science, Chulalongkorn University, Thailand
- Faculty of Information and Technology, Zhejiang Chinese Medical University

HOME ADDRESS Building A, no 779/110, Regent Home Sukhumvit 81 Khwaeng Suan Luang, Khet Suan Luang, Krung Thep Maha Nakhon, Bangkok, Thailand.

PUBLICATION Hangyi Wang and Chidchanok Lursinsap. CRYPTOSYSTEM FOR TEXTUAL MESSAGE BY USING MALLEABLE NEURAL NETWORK WITH SECRET DIMENSIONS. In International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology, IEEE, 2021.

AWARD RECEIVED - 2015 Fifth Level/Primary Skill Level of Western-style cook by Seal of Occupational Skill Testing Authority, The Ministry of Human Resources and Social Security of People's Republic of China. Certificate No. 1511010000504526.
- 2015 Primary Skill Level of Elementary Chinese Sign Language in Zhejiang, China.
-2016 Third Level/Senior Skill Level of dietitian by Seal of Occupational Skill Testing Authority, The Ministry of Human Resources and Social Security of People's Republic of China. Certificate No. 1611000000338379.
- 2019, 2020 Teaching Assistant Scholarship by Chulalongkorn University, Thailand.