



Instant-Hybrid Neural-Cryptography (IHNC) based on fast machine learning

Assem Badr¹

Received: 20 October 2021 / Accepted: 13 June 2022 / Published online: 13 July 2022
© The Author(s) 2022

Abstract

Nowadays, cryptographic systems' designers are facing significant challenges in their designs. They have to constantly search for new ideas of fast unbreakable algorithms with a very powerful key generator. In this paper, we propose a novel hybrid neural-cryptography methodology. It depends on new rule of very fast Backpropagation (BP) instant machine learning (ML). This proposed Hybrid Cryptography system is constructed from Encryptor and Decryptor based on the asymmetric Autoencoder type. The Encryptor encrypts and compresses a set of data to be instant code (i-code) using public key. While the Decryptor recovers this i-code (ciphered-data) based on two keys together. The first is the private key and the other is called instant-key (i-key). This i-key is generated from 3 factors as well (the original data itself, the generated i-code and the private key). The i-key is changing periodically with every transformation of plain data set, so it is powerful unpredictable key against the brute force.

Keywords Hybrid encryption · Neural-cryptography · Machine learning · Learning ratio ($\Delta\ell$)

1 Introduction

Nowadays, information security has become a very important issue for governments, companies and even individuals. Therefore, cryptographic systems' designers must constantly search for up-to-date cryptography algorithms with higher efficiency to counter these potential threats.

Usually, any cryptographic security level depends on two major complicating factors. The first factor depends on the complexity of the selected Cryptographic Algorithm (CA) that responsible to convert plaintext to ciphertext, or vice versa. The second factor is the complexity of the secret key(s) generation. Both factors must be difficult enough that the information cannot be accessed or predicted easily.

There are two main types of CA. The first type is called Asymmetric Key Algorithm (AKA) or Public-key

cryptography while the other is called Symmetric Key Algorithm (SKA).

Essentially, any security system are built on the basis of CA with its Cryptographic Keys (CK). These keys should be known only to the main users and hidden from others. Attackers often try to find, explore, extract, or even steal CK to recover hidden and confidential information. Significantly, as long as the CK is confidential, the information is safe.

In practice, keeping CK secret is one of the most difficult problems. Therefore, CK must be very complex (very long) to be retrieved. Thus, the CK are generated using powerful algorithms to ensure that each CK is unpredictable [1].

There is a direct proportion between the n-bits of CK and its security level. Oftentimes, n-bit security means that attackers have up to 2^n times to break it [2].

Towards more powerful CA and CK, designers thought to integrate and collect more than one CA (often AKA and SKA) in single algorithm called Hybrid cryptography technique. So, this combination of multiple encryption rules simultaneously takes advantage of their strengths. They collected as many strengths as possible. Besides, they also get rid of many weaknesses as possible.

✉ Assem Badr
abmageed@eng.modern-academy.edu.eg

¹ Computer Department, Modern Academy for Engineering and Technology, Cairo, Egypt

Hybrid encryption is a unique technology that combines a strong algorithm (such as AKA encryption) with a low execution time (likes SKA encryption). It utilizes more than one key to perform its task, each key has long width (Larger number of bits) likes AKA. Moreover, it handles large amount of data simultaneously with highly speed of processing likes SKA.

The essential advantage of the hybrid encryption is that it generates a new encrypted key with each new transformation from plain to cipher data [3]. Frequently, this key is generated randomly by strong algorithms. According to this power and the constant changes of the generated key, hackers can't anticipate it. Therefore, this key is the second line of defense against hackers. Moreover, this generated key ensures stronger security during data transmission over the communication media, because this key is unpredictable.

Toward strongest hybrid cryptography, designers utilized neural network (NN) in the field of cryptography that called Neuro-Cryptography (or neural cryptography). This concept is now rapidly increasing. In the past two decades, many researchers already have collected and combined different NNs with various classical cryptographic paradigms [4]. The designers invented many complex neuro-cryptography rules [5–7]. They have created their CA by using different learning algorithms to generate strong keys [8–12].

As long as the NN architecture (within neural cryptography) is deeper, the generated keys are too complex to predict.

In recent years, machine learning (ML) techniques have become increasingly powerful in cryptography while processing more than 3 quintillion (3 followed by 18 zeros) of data-bytes around the world every day [13]. ML techniques can be used to obtain the relationship(s) between original data and its encrypted data within cryptographic systems. Also, it can be used to generate CK. Furthermore, ML is used to compress/decompress messages before they are encrypted/decrypted.

The concept of utilizing different machine learning (ML) in Neuro-Cryptography is growing rapidly. Various researchers have proposed many Neuro-Cryptography rules [14–16].

From literature survey, there is a lack of research on hybrid-type Neuro-Cryptography (particularly for autoencoders) using backpropagation ML. Therefore, this research proposes a novel hybrid neuro-cryptography rule with instant-changing key based on the fast BP ML paradigm called “Instant Learning-Ratio Machine-Learning (ILRML)” [17].

Due to the great dependence of the ILRML algorithm for our proposed Neuro-Cryptography design, we will dedicate a large space to it in this introduction section.

As its name implies, the ILRML is very fast BP supervised learning. It can update all the weights in the NN during a single iteration. So, it can be used to encrypt/decrypt the original/ciphered data during the online communications. It relies on the learning ratio (Δl) rather than the learning rate (η). In spite the ILRML runs its forward propagation (FP) like any conventional ML rule, but it has a fundamental difference during BP. The Fig. 1 shows the concept of ILRML for one neuron.

For simple explanation of ILRML, we will firstly discuss the case of single neuron within a NN. The ILRML checks (each learning iteration) the difference between any instant output from neuron and its instant desired target (d). If that difference is unacceptable (according to the required accuracy), then the ILRML enables intersections between the curve of this neuron's activation function and the constant function of d as shown in the Fig. 2.

The intersection may be one point (or two points due to the curvature of activation function). In our case, the constant function $d = 0.8$ intersect with the functions (Linear, Sigmoid, Tanh and ReLU) at the points (2.0, 1.3863, 1.0986 and 0.8) respectively. These intersection points are called (post-intersection). It considered as instant updated pre-activation factor and symbolled by x^{new} .

The ILRML divides the pre-activation factor x^{new} (of the assigned function) over the old pre-activation factor (the old sum of product) x^{old} to generate the instant learning ratio (Δl) of this neuron during that iteration.

The ILRML uses the factor Δl to update the old values of the neuron's inputs and their weights by multiplying them according to the next algorithms 1 and 2.

In fact, the factor x^{new} can be determined (in the programming of this algorithm) from the inverse of the activation function (at the point d).

$$x^{\text{new}} = \text{inverse}(\text{activationfunction})|_d \quad (1)$$

As we mentioned before, the ILRML differs in their BP. So, we will discuss two associated algorithms from BP point-of-view. The algorithm 1 explains the feed-backward during BP from a neuron (in any layer) to all neurons in the previous layer as demonstrated in Fig. 3 [17]. While the algorithm 2 illustrates the feed-backward from all neurons (of any layer) to one neuron in the previous layer as demonstrated in Fig. 4 [17].

The remaining relevant issues in this research will be organized in this way. In Sect. 2, the description of the basic architecture of the IHNC system and its general key

Fig. 1 The concept of ML based on the learning ratio within a neuron

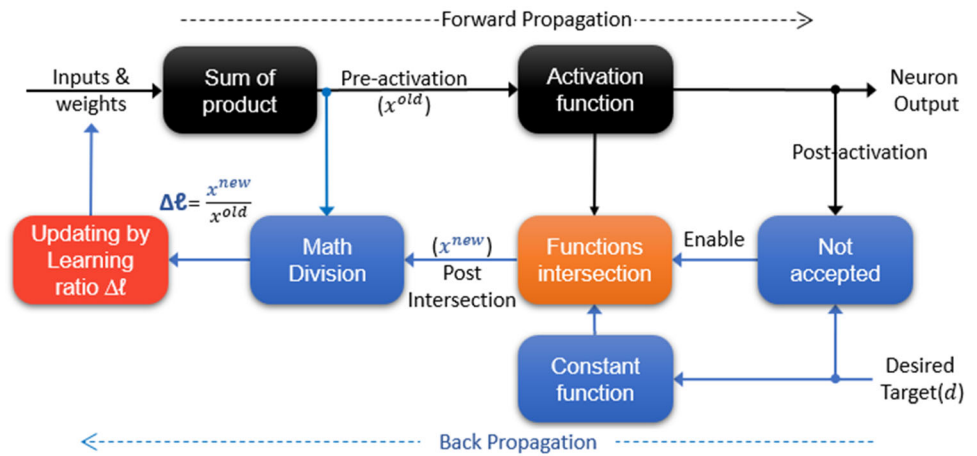


Fig. 2 Intersections between the desired target ($d = 0.8$) and many activation functions

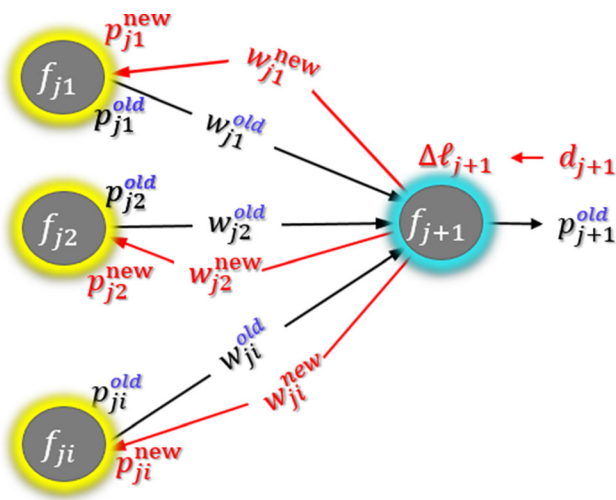
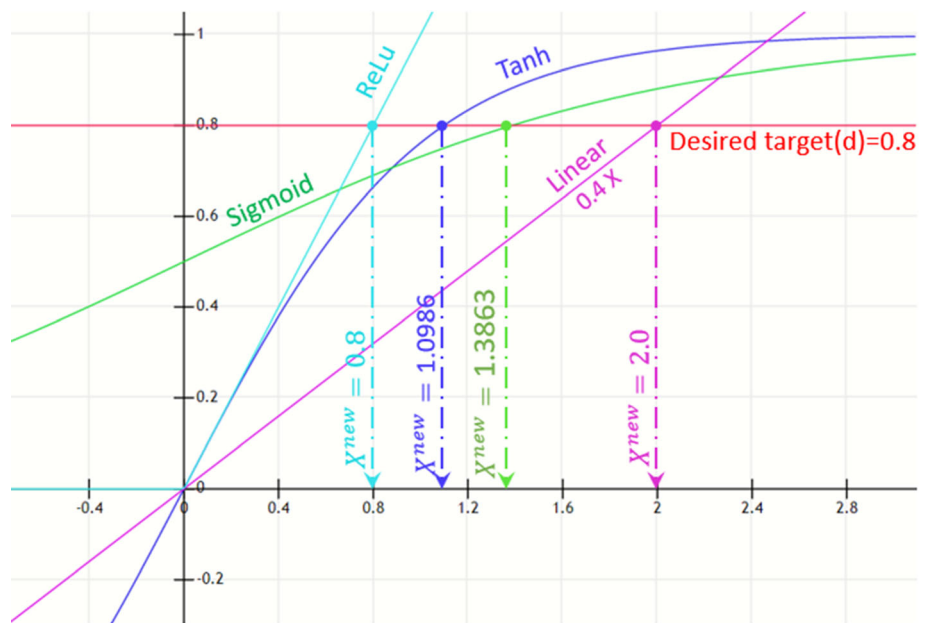


Fig. 3 The BP from a neuron to the previous neurons

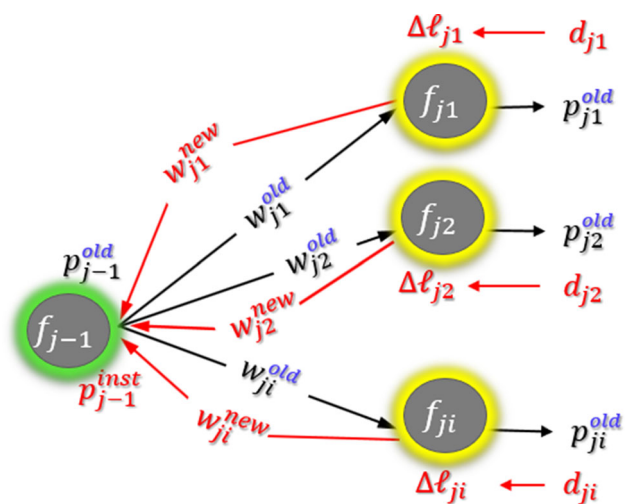


Fig. 4 The BP from all neurons (within a layer) to a previous neuron

formats besides its workflow. In Sect. 3, proposes an architecture of a lite-IHNC. It includes 2 sub-sections. The first for describing its Encryptor-unit and the second for its

Decryptor-unit. The Sect. 4, includes proposal of a Deep-IHNC (to confirm our idea) as a case study with its results. Eventually, the conclusion and future hope.

Algorithm (1): Running the ILRML to update the parameters of all neurons (i) in the layer (j) those fed backward from a neuron in layer ($j + 1$)

1. Start instant training
2. Start Forward-Propagation by receiving all fires ($p_{j_1}^{old}, p_{j_2}^{old}, \dots, p_{j_i}^{old}$) from layer (j) via their weights ($w_{j_1}^{old}, w_{j_2}^{old}, \dots, w_{j_i}^{old}$) to a neuron in the layer ($j + 1$).
3. Determine old Pre-activations ($x_{j_1}^{old}, \dots, x_{j_i}^{old}$) & old post-activation (p_{j+1}^{old}) of the neuron ($j + 1$).

$$p_{j+1}^{old} = f_{j+1}(X_{j+1}^{old}) = \sum_i w_{j_i}^{old} * p_{j_i}^{old}$$

4. Receive the desired target as constant function ($f_{constant} = d_{j+1}$) to the neuron in layer ($j + 1$).
5. If ($p_{j+1}^{old} - d_{j+1}$) is not acceptable then do the next
6. Start Backpropagation by intersecting (\cap) the activation-function f_{j+1} with the constant function ($f_{constant}$) to get new instant Pre-activation (X_{j+1}^{new})

$$f_{j+1} \cap f_{constant} \xrightarrow{yield} X_{j+1}^{new}$$

7. Get new *instant* learning ratio ($\Delta\ell_{j+1}$) = $X_{j+1}^{new} / X_{j+1}^{old}$ by dividing the new instant Pre-activation (X_{j+1}^{new}) over the old Pre-activation (X_{j+1}^{old}).
8. If the layer (j) is an input layer then update only its weights by using the new instant learning ratio ($\Delta\ell_{j+1}$) as follows

$$w_{j_i}^{new} = \Delta\ell_{j+1} * w_{j_i}^{old}$$

9. If the layer (j) is either hidden or output layer then update its old weights ($w_{j_1}^{old}, w_{j_2}^{old}, \dots, w_{j_i}^{old}$) besides its old fires ($p_{j_1}^{old}, p_{j_2}^{old}, \dots, p_{j_i}^{old}$) to get all desired targets ($d_{j_1}^{new}, d_{j_2}^{new}, \dots, d_{j_i}^{new}$) for all neurons in layer (j) as follows

$$d_{j_i}^{new} = p_{j_i}^{old} \sqrt{|\Delta\ell_{j+1}|}$$

$$w_{j_i}^{new} = w_{j_i}^{old} \sqrt{|\Delta\ell_{j+1}|}$$

Algorithm 1 The BP of the ILRML from a neuron to the previous layer.

Algorithm (2): Running the ILRML to update the parameters of a neuron in layer ($j - 1$) that fed backward from all neurons (i) in layer (j)

1. Start instant training
2. A neuron fires its output (p_{j-1}^{old}) toward multiple neurons (i) in a layer (j) through their weights ($w_{j1}^{old}, w_{j2}^{old}, \dots, w_{ji}^{old}$).
3. Start forward propagation to get the old Pre-activations ($x_{j1}^{old}, \dots, x_{ji}^{old}$) & the Post-activations ($p_{j1}^{old}, \dots, p_{ji}^{old}$) of all neurons (i).

$$p_{ji}^{old} = f(x_{ji}^{old}) = w_{ji}^{old} * p_{j-1}^{old}$$
4. Receive the desired targets (d_{ji}) to each neuron (i) in the layer (j).
5. If any difference ($d_{ji} - p_{ji}^{old}$) not acceptable of any neuron (i) then do the next
6. Start backpropagation to get all new instant Pre-activations ($x_{j1}^{new}, \dots, x_{ji}^{new}$) by intersect (\cap) each neuron's Activation-function with their Constant-functions of the desired targets (d_{ji}).

$$(f_{activation})_{ji} \cap d_{ji} \xrightarrow{yield} x_{ji}^{new}$$
7. Get new instant learning ratios ($\Delta\ell_{ji}$) for each neuron (i) in a layer (j).

$$\Delta\ell_{ji} = \frac{x_{ji}^{new}}{x_{ji}^{old}}$$
8. Update the weights between layer (j) and layer ($j - 1$) and their feedback as follows:

$$w_{j1}^{new} = w_{j1}^{old} * \frac{\Delta\ell_{j1}}{\sqrt{|\Delta\ell_{j1}| |\Delta\ell_{j2}| \dots |\Delta\ell_{ji}|}}$$

$$w_{ji}^{new} = w_{ji}^{old} * \frac{\Delta\ell_{ji}}{\sqrt{\prod_t |\Delta\ell_{ji}|}}$$
9. Update the old fire (p_{j-1}^{old}) to get new instant desired target (d_{j-1}^{new}) for a neuron in layer ($j - 1$)

$$d_{j-1}^{new} = d_{j-1}^{inst} = p_{j-1}^{old} * \sqrt{|\Delta\ell_{j1}| |\Delta\ell_{j2}| \dots |\Delta\ell_{ji}|} = p_{j-1}^{old} * \sqrt{\prod_t |\Delta\ell_{ji}|}$$

Algorithm 2 The ILRML BP from all neurons of a layer to one of previous neuron.

2 The basic architecture of the IHNC

In this section, we will introduce general architecture of our proposed IHNC. Let's first discuss three major requirements for this design. The first requirement is to find a suitable structure of an Artificial Neural Network (ANN) that can be divided into 2 complementary sub-ANNs to perform two tasks (encryption task and decryption task).

The second requirement, the number of outputs (from the first divided sub-ANN) must equal the same number of inputs (to the second sub-ANN) to transfer the ciphered-data between them.

Third requirement, the selected ANN should have similar numbers of its inputs and outputs to recover the same encrypted (original) data.

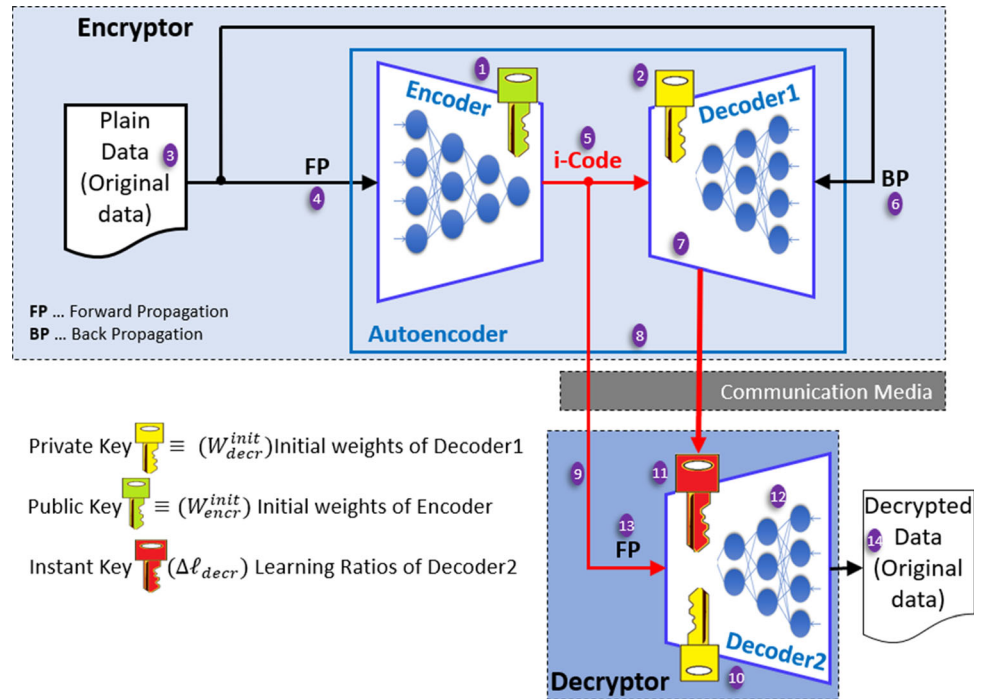
According to the previous requirements, neural autoencoders are the best choice for our construction.

The neural autoencoder has two conjugated sub-ANNs (Encoder and Decoder) [18–20].

Therefore, we will utilize these two sub-ANNs (autoencoder stages) to carry out our idea as will be explained later on.

The main structure of the IHNC system should contain two units (Encryptor and Decryptor) as seen in the Fig. 5. The Encryptor-unit includes a fully autoencoder (Encoder and Decoder stages) while the Decryptor contains only its Decoder-stage.

Fig. 5 The basic architecture of the proposed IHNC (Encryptor and Decryptor)



All initial values of the Encoder-stage are considered as the public-key (green key) of the IHNC. Moreover, all the initial values of the decoder-stage (decoder1 or decoder2) are considered as the private key (yellow key) of the IHNC.

The Encryptor-unit carries out 2 tasks. The first task is relying on the FP by passing an instant sample of plain data (original data) through the initial weights (Public key) of encoder-stage only to generate an instant code (i-Code). This i-Code represents the instant ciphered data. Moreover, it can also call the compressed code according to the autoencoder. The second task (of the Encryptor) depends on the BP by passing the same sample of plain data through the initial weights (Private-key) of the Decoder1. This second task generates Learning ratios (as instant-key) from all neurons in this decoder. Both the generated instant-key (i-key) and instant-code (i-Code) are changing periodically with each new sample of plain-data to the Encryptor-unit.

Both i-key and i-code are sent to the Decryptor-unit (directly or through suitable communication media).

On the other hand, the Decryptor-unit includes only decoder-stage (Decoder2). It exactly similar to the Decoder1 in the Encryptor-unit. The Decoder1 is loaded by the private key (initial values of its weight) and it receives both the i-Code and i-key to recover the instant original data.

As mentioned before, we have Public, Private and Instant keys. The Public and Private keys consist of all initial values of the weights within Encoder-stage and

Decoder-stage respectively. While the i-key consists of all generated learning ratios ($\Delta \ell$) from Decoder1.

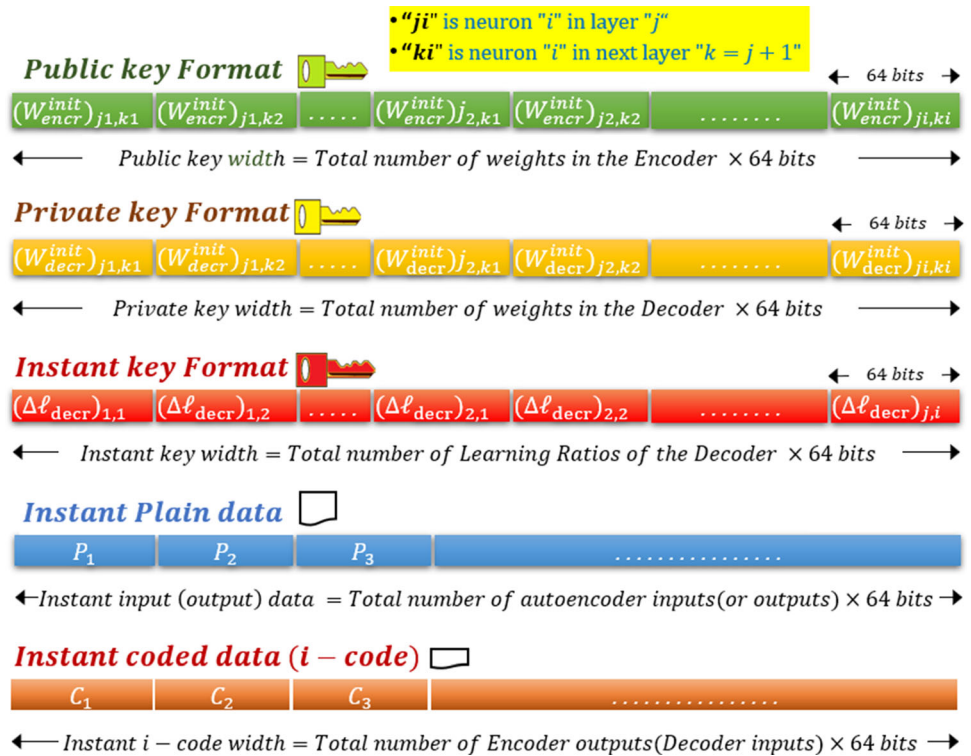
Like any computer system, all weights values are represented by the Double-precision floating-point (Flot64) format to get high accuracy numbering during the FP and BP calculations through the neural autoencoder.

During the IHNC design, the designers are free to choose any number of neurons inside their autoencoders, and then any number of their weights. Therefore, the size of their keys can vary from one design to another (according the security level requirement). This gives flexibility in the designs because the size of their keys is not fixed like many recent encryption systems. let us give examples. If the public or private key contains only 10 weights, then the size of that key will be 640 bits while if it contains 1000 weights, then it will be 64,000 bits and so on.

Also, the instant key depends on the number of neurons (not the weights) inside the decoder1. For example, if it contains 100 neurons, the i-key size will be 6400 bits.

Further, IHNC designers are free to arrange those weights in concatenation form. For example, as shown in Fig. 6. Here the designer has arranged all the weights of the two keys (public and private) in a regular order. The designer started with the first weight of the first neuron of the Encoder-stage “ $(W_{encl}^{init})_{j1,k1}$ ” and then the second weight of the same neuron “ $(W_{encl}^{init})_{j1,k2}$ ” until he finishes all the weights of the first neuron. Then, he completes the rest of the weights in the same way.

Fig. 6 Bit-format of IHNC parameters (Public, Private, Instant, i-code and Plain data)



Also, he arranged all learning ratios from the first neuron “ $(\Delta_{decr}^l)_{1,1}$ ” to the last one “ $(\Delta_{decr}^l)_{j,i}$ ” inside the i-key.

Moreover, Fig. 6 shows the width of the instant plain-data (total number of either autoencoder inputs or output * 64 bits). Besides, it shows the width of the instant compressed data or i-code that will be transferred from the Encryptor to Decryptor (total number of either Decoder-stage inputs or Encoder-stage outputs * 64 bits).

Furthermore, the designers are more free if they rely on the principle of Asymmetric Stacked Autoencoder (ASA) characteristics [21], they are free to choose how many layers will be inside the encoder-stage and the rest of the asymmetric-autoencoder layers to the decoder unit.

This methodology gives more flexibility to their designs because the same IHNC autoencoder can decrypt the same plain-data in many i-codes and i-keys at the same time. In addition, send them to several users (those have the compatible decoders) at once.

In this case, subscribers will receive the same information, but with different encrypted codes and instant keys. Therefore, they can retrieve the same original information with different keys those agreed between the sender and all the recipients.

For instance, assume IHNC application has an ASA with seven layers as seen in the Fig. 7. It can send three i-codes (i-code1, i-code2 and i-code3) for the same plain-data simultaneously from three encoders (Encoder-a, Encoder-b and Encoder-c) respectively.

Now let’s explain the whole operations of the IHNC system through two flowcharts. The first one for carrying out the encryption as demonstrated in the Fig. 8. In step-1, the user uploads the public-key (as initial weights) to the Encoder in the encryptor-unit. Step-2, the user loads the private-key (initial weights) in the Decoder1 of the encryptor too. From Step-3 to step-5, the user loads his/her instant plain-data to the input of the encryptor-unit and run the FP to all neurons in the encoder. So they get the instant ciphered-data or instant-code (i-code) from the Encoder-stage of the autoencoder. In step-6 and step-7, The user feeds backward the same samples of the instant inputs to the Decoder-stage (as desired target) and run the instant learning ratios BP to Decoder1 only. So the user get the instant-key (i-key) as the learning ratios (Δ^l) of all neurons within Decoder1. In the last step-8, both the i-key and i-code are sending to the decryptor-unit.

On the other side, another user receives both i-code and i-key to the compatible Decryptor (Decoder2) as step-9 and step-11 respectively in Fig. 9. Also the user loads the common private-key as step 10. In step-12 the Encryptor-unit updates the received i-code and initial weights of Decoder using the received i-key. In the last 2 steps (13 and 14), the Decryptor run its FP to recover the instant original-data as will be explained in the next section.

Fig. 7 An ASA generates several i-codes from multiple encoders simultaneously

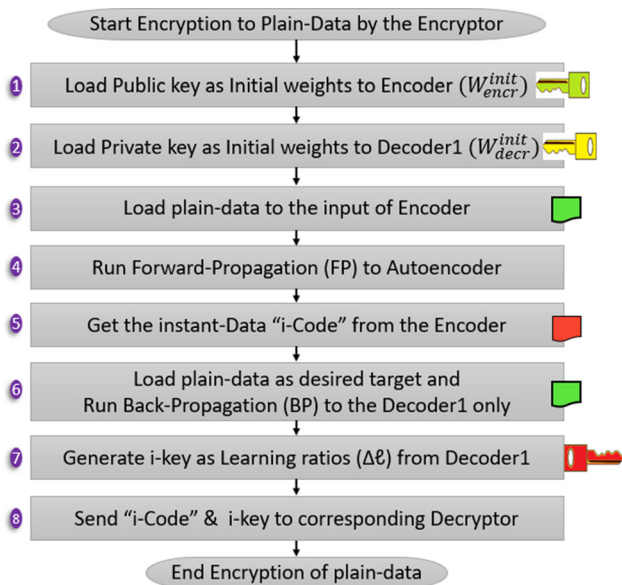
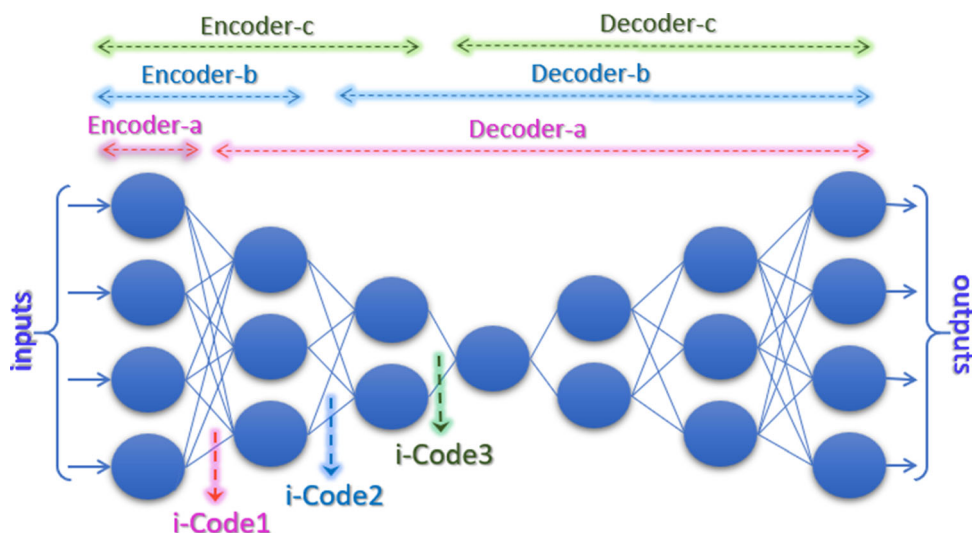


Fig. 8 The workflow of the encryption process of the basic IHNC

3 Architecture and methodology of a lite-IHNC

In this section, we will discuss a simple structure of IHNC system called lite-IHNC. It operates depending on the ML algorithms of the instant $\Delta\ell$ that discussed in introduction-section.

First, before we start diving into the details (formulas, algorithms, applications), we have to provide the Symbol-key for the next part of this research. Therefore, assume an ANN includes two subsequent layers "j" and layer "k" ($k = j + 1$) as seen in Fig. 10. Each layer has "i"-number of neurons. The weights between them are labeled by two parameters. The First is the number of the source neuron and the other is the number of the destination neuron. For

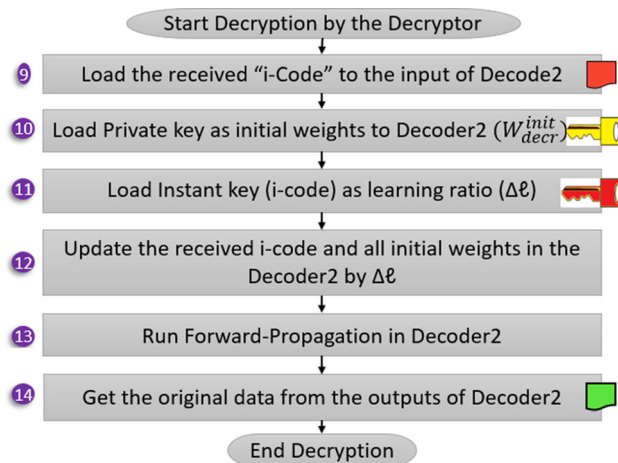


Fig. 9 The workflow of decryption process of the basic IHNC

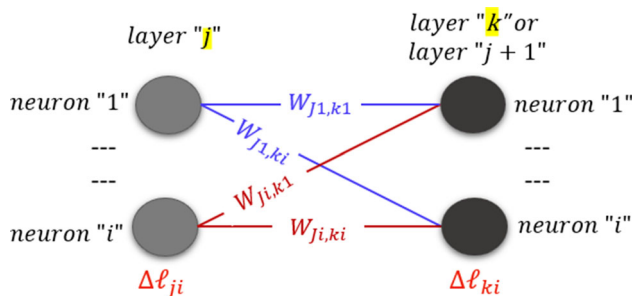


Fig. 10 A simple ANN indicates the used symbols in the rest of this research

instance, " $W_{j1,ki}$ " indicates the weight between the first neuron of layer "j" and the "i" neuron in the destination layer "k". Moreover, the symbol $\Delta\ell_{k8}$ indicates the learning ratio of eighth neuron in the layer "k" and so on.

Now, all details of the lite-IHNC will be discussed in the next sub-sections.

3.1 Encryptor-unit of the lite-IHNC

This subsection is assigned to present the main architecture of the Encryptor within the proposed lite-IHNC. It consists of 3 layers as illustrated in the Fig. 11. Its encoder-stage includes two layers (input-layer “j” and layer “c”). The layer “j” has 8 inputs ($p_1 : p_8$) via 8 weights ($w_{p1,j1} : w_{p8,j8}$). The layer “j” is connected with layer “c” via 8 weights ($w_{j1,c1} : w_{j4,c1}$ and $w_{j5,c2} : w_{j8,c2}$).

All weights in that encoder-stage represents the public-key (with size 1024 bits) as shown in the upper segment of the Fig. 12 and step (1) of algorithm 3.

While, the decoder-stage (in this Encryptor) contains only one layer called “op”. It connected with layer “c” via 16 weights ($w_{c1,op1} : w_{c1,op8}$ and $w_{c2,op1} : w_{c2,op2}$). These weights represent the private-key (width 1024 bits) as the lower segment in the Fig. 12 and step (2) of algorithm 3.

An instant data sample is delivered to the encoder inputs ($p_1 : p_8$) as step (3) of algorithm (3).

The FP runs inside the Encryptor-unit to generate the i-codes ($C1^{old}$ and $C2^{old}$) as step (4) of algorithm 3 and as illustrated in the formulas (2: 6). Moreover, it generates the

sum-of-products that called “*Pre_activation^{old}*” (from decoder stage) as demonstrated in formula (7).

$$C1^{old} = f[W_{J1,C1} * f(j1) + W_{J2,C1} * f(j2) + W_{J3,C1} * f(j3) + W_{J4,C1} * f(j4)] \tag{2}$$

$$C1^{old} = f[W_{J1,C1} * f(W_{p1,j1} * P_1) + W_{J2,C1} * f(W_{p2,j2} * P_2) + W_{J3,C1} * f(W_{p3,j3} * P_3) + W_{J4,C1} * f(W_{p4,j4} * P_4)] \tag{3}$$

$$C2^{old} = f[W_{J5,C2} * f(j5) + W_{J6,C2} * f(j6) + W_{J7,C2} * f(j7) + W_{J8,C2} * f(j8)] \tag{4}$$

$$C2^{old} = f[W_{J5,C2} * f(W_{p5,j5} * P_5) + W_{J6,C2} * f(W_{p6,j6} * P_6) + W_{J7,C2} * f(W_{p7,j7} * P_7) + W_{J8,C2} * f(W_{p8,j8} * P_8)] \tag{5}$$

$$i_code = \begin{bmatrix} C1^{old} \\ C2^{old} \end{bmatrix} \tag{6}$$

Fig. 11 The architecture of the Encryptor of the lite-IHNC

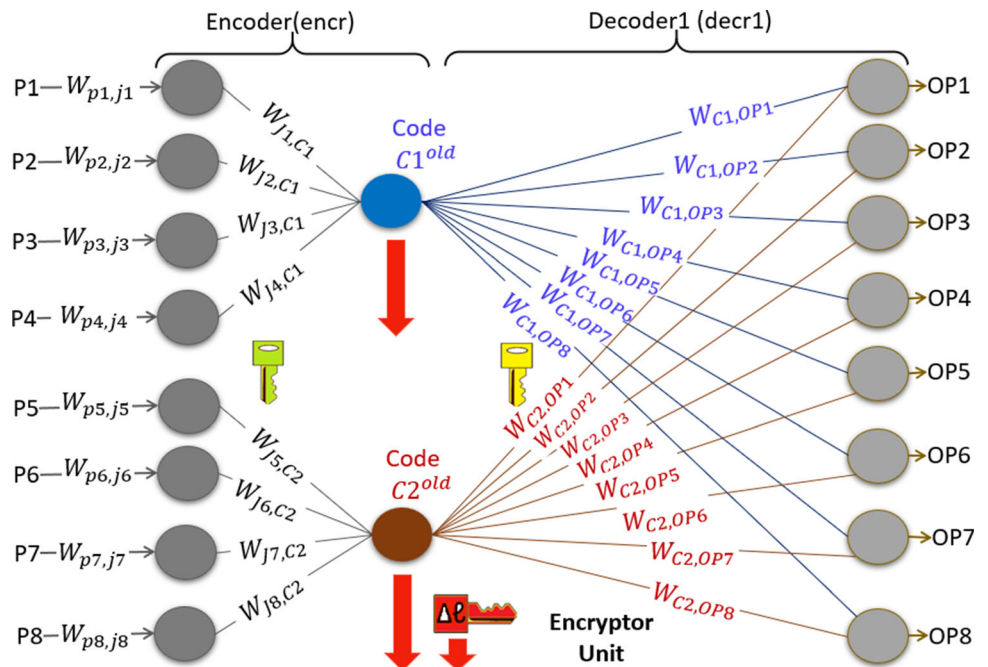
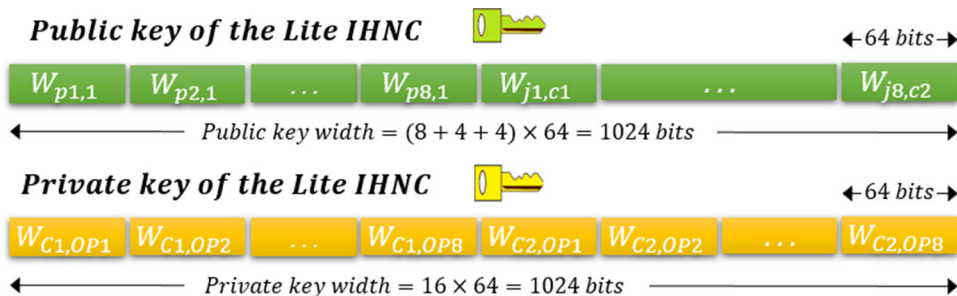


Fig. 12 The bit format of the private and public keys of the lite-IHNC



$$\begin{aligned}
 \text{Pre_activation}^{old} &= \begin{bmatrix} \text{OP1} \\ \text{OP2} \\ \text{OP3} \\ \text{OP4} \\ \text{OP5} \\ \text{OP6} \\ \text{OP7} \\ \text{OP8} \end{bmatrix} \\
 &= \begin{bmatrix} W_{C1,op1}^{init} & W_{C2,op1}^{init} \\ W_{C1,op2}^{init} & W_{C2,op2}^{init} \\ W_{C1,op3}^{init} & W_{C2,op3}^{init} \\ W_{C1,op4}^{init} & W_{C2,op4}^{init} \\ W_{C1,op5}^{init} & W_{C2,op5}^{init} \\ W_{C1,op6}^{init} & W_{C2,op6}^{init} \\ W_{C1,op7}^{init} & W_{C2,op7}^{init} \\ W_{C1,op8}^{init} & W_{C2,op8}^{init} \end{bmatrix} * \begin{bmatrix} C1^{old} \\ C2^{old} \end{bmatrix} \\
 &= \begin{bmatrix} W_{C1,op1}^{init} * C1^{old} & W_{C2,op1}^{init} * C2^{old} \\ W_{C1,op2}^{init} * C1^{old} & W_{C2,op2}^{init} * C2^{old} \\ W_{C1,op3}^{init} * C1^{old} & W_{C2,op3}^{init} * C2^{old} \\ W_{C1,op4}^{init} * C1^{old} & W_{C2,op4}^{init} * C2^{old} \\ W_{C1,op5}^{init} * C1^{old} & W_{C2,op5}^{init} * C2^{old} \\ W_{C1,op6}^{init} * C1^{old} & W_{C2,op6}^{init} * C2^{old} \\ W_{C1,op7}^{init} * C1^{old} & W_{C2,op7}^{init} * C2^{old} \\ W_{C1,op8}^{init} * C1^{old} & W_{C2,op8}^{init} * C2^{old} \end{bmatrix} \tag{7}
 \end{aligned}$$

During the BP (for the decoder-stage only), the same data sample is delivered also to the decoder as desired target “ p_i ” as step (5) of same algorithm 3. According to the instant-ML of the learning ratios (that introduced in introduction part), each desired target intersects with its corresponding neuron function-curve to produce new “ $Pre_activation^{new}$ ” for each neuron “ i ” of the output-layer “ op ” as formula (8 and 9).

These intersections are similar as carrying out the neuron inverse-function (as mentioned earlier in the formula (1) above in the introduction part).

$$\text{Pre_activation}^{new}(i) = \text{Pre_activation}^{Desired}(i) = f^{-1}(p_i) \tag{8}$$

$$\begin{aligned}
 \text{Pre_activation}^{new} &= \begin{bmatrix} \text{Pre_activation}^{Desired}(1) \\ \text{Pre_activation}^{Desired}(2) \\ \text{Pre_activation}^{Desired}(3) \\ \text{Pre_activation}^{Desired}(4) \\ \text{Pre_activation}^{Desired}(5) \\ \text{Pre_activation}^{Desired}(6) \\ \text{Pre_activation}^{Desired}(7) \\ \text{Pre_activation}^{Desired}(8) \end{bmatrix} \\
 &= \begin{bmatrix} f^{-1}(p1) \\ f^{-1}(p2) \\ f^{-1}(p3) \\ f^{-1}(p4) \\ f^{-1}(p5) \\ f^{-1}(p6) \\ f^{-1}(p7) \\ f^{-1}(p8) \end{bmatrix} \tag{9}
 \end{aligned}$$

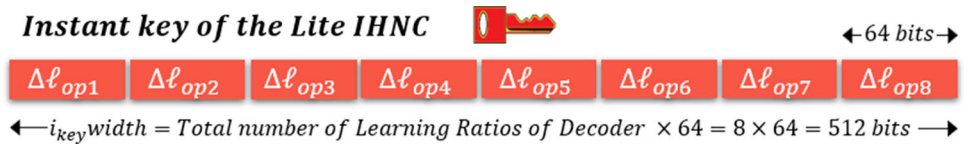
Each new pre-activation “ $Pre_activation^{Desired}(i)$ ” value will be divided over its corresponding old-activation value to produce its learning ratios(i) as formula (10) and step (6) of the same algorithm. In this case, all generated learning ratios ($\Delta_1 : \Delta_8$) will be considered as the generated i-key. Thus here, the i-key size is $8\Delta_s * 64 \text{ bit} = 512 \text{ bits}$ as illustrated in the Fig. 13.

These generated i-key and i-code will be transferred to the Decryptor-unit as step (7 and 8) of the same algorithm.

$$\begin{aligned}
 \Delta \ell_i &= \frac{\text{Pre_activation}^{Desired}(i)}{\text{Pre_activation}^{old}(i)} \\
 &= \frac{f^{-1}(p_i)}{\text{Sum of products of neuron } (op_i)} \tag{10}
 \end{aligned}$$

Algorithm 3 the encryption process of the lite-IHNC.

Fig. 13 The i-key bit-format of the lite-IHNC



Algorithm(3): Encryption steps of the Instant Hybrid Neuro-Cryptography (IHNC) based on "Δℓ" for the proposed Encryptor of the figure (11)

1. Load Public key ($W_{p1,j1}$ to $W_{p8,j8}$) and ($W_{j1,c1}$ to $W_{j8,c2}$) as initial values of the weights in the Encoder-stage of the Autoencoder.
2. Load Private key ($W_{c1,op1}$ to $W_{c1,op8}$) and ($W_{c2,op1}$ to $W_{c2,op8}$) as initial values of the weights in the Decoder-stage of the Autoencoder.
3. Load sample of eight plain data ($P_1: P_8$) into the input-layer "j" of the Autoencoder.

----- Instant Training of Encryption -----

4. Run Forward Propagation for the Encoder-stage to generate the old values of the Codes ($C1^{old}$) & ($C2^{old}$).
5. Load the same sample of plain data into the output-layer of the Decoder-stage as desired targets.
6. Run Backpropagation for the Decoder-stage only based on the Instant-Learning-Ratios Machine Learning rule (ILR-ML) to get the Learning ratios ($\Delta\ell_1: \Delta\ell_8$) as formula (10) that represent the Instant Key.

----- Sending the ciphered data & Instant key to the Encryptor -----

7. Send all Codes ($C1^{old}$) & ($C2^{old}$) as ciphered-data.
8. Send all Instant Learning ratios ($\Delta\ell_1: \Delta\ell_8$) as Instant Key.

3.2 Decryptor-unit of the lite-IHNC

This subsection presents the main architecture of the Decryptor within the lite-IHNC. It includes only decoder-stage (Decoder2) which is exactly the same as decoder1 in the Encryptor-unit as demonstrated in the Fig. 14. It loads by the initial values (private key) as step (1) of algorithm 4. Also, it receives the previously generated i-key ($\Delta_1 : \Delta_8$) as step (2) of algorithm 4. Besides, it receives the i-code as step (3) of algorithm 4. It will carry out the following two points.

Firstly, it updates the received i-codes ($C1^{old}$ and $C2^{old}$) according to the formula of the step (9) in the algorithm 2 to be ($C1^{new}$ and $C2^{new}$) as illustrated in the formulas (11 and 12) and step (5) of algorithm 4.

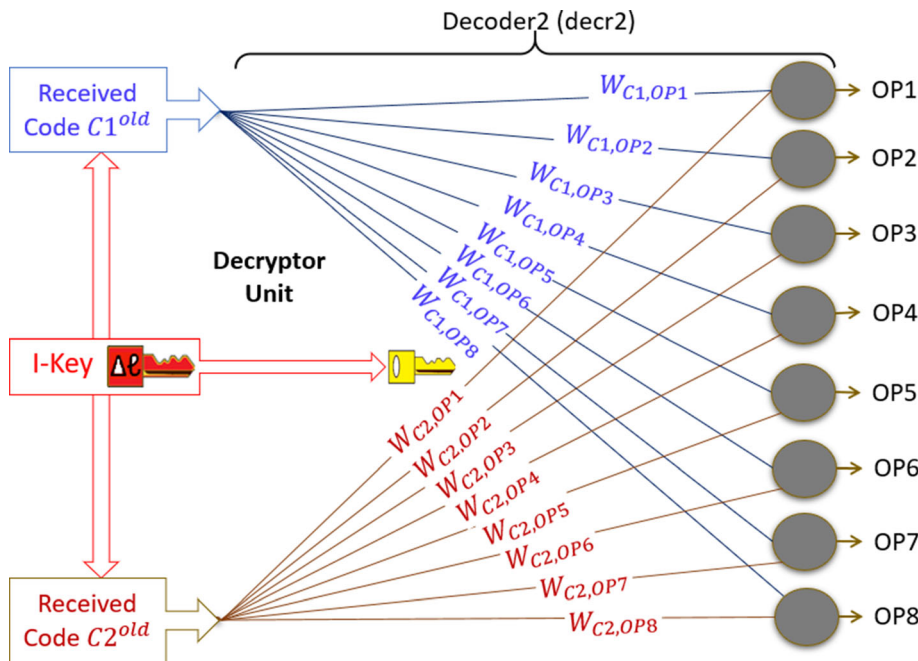
$$C1^{new} = C1^{old} * \sqrt{\prod_{i=1}^8 |\Delta\ell_i|} \tag{11}$$

$$C2^{new} = C2^{old} * \sqrt{\prod_{i=1}^8 |\Delta\ell_i|} \tag{12}$$

Secondly, all Decoder's weights are updated according to the formula in the step (8) in the algorithm 2. Here, only 2 formulas are written ($W_{C1,op1}^{new}$ and $W_{C2,op1}^{new}$) for the neuron-1as illustrated in the formulas (13 and 14) and step (4) of algorithm 4.

$$W_{C1,op1}^{new} = (W_{C1,op1}^{old} * \Delta\ell_1) / \sqrt{\prod_{i=1}^8 |\Delta\ell_i|} \tag{13}$$

Fig. 14 The architecture of the Decryptor of the lite-IHNC



$$W_{C2,op1}^{new} = (W_{C2,op1}^{old} * \Delta l_1) / \sqrt{\prod_{i=1}^8 |\Delta l_i|} \tag{14}$$

after that, the Decryptor unit needs only the FP to recover the original data from the Decoder2. One sample of this recovered data from the 1st neuron is demonstrated in formula (15).

$$OP1^{new} = f(C1^{new} * W_{C1,op1}^{new} + C2^{new} * W_{C2,op1}^{new}) \tag{15}$$

By substituting the formulas (11, 12, 13, 14) in formula (15) will get formula (16) thus (17).

$$OP1^{new} = f \left(C1^{old} * \sqrt{\prod_{i=1}^8 |\Delta l_i|} * W_{C1,op1}^{old} * \frac{\Delta l_1}{\sqrt{\prod_{i=1}^8 |\Delta l_i|}} + C2^{old} * \sqrt{\prod_{i=1}^8 |\Delta l_i|} * W_{C2,op1}^{old} * \frac{\Delta l_1}{\sqrt{\prod_{i=1}^8 |\Delta l_i|}} \right) \tag{16}$$

$$OP1^{new} = f[\Delta l_1 * (C1^{old} * W_{C1,op1}^{old} + C2^{old} * W_{C2,op1}^{old})] \tag{17}$$

From formula (10), will get the learning ratio (Δl_1) of the 1st neuron as seen in the formula (18). And so on, for all rest neurons' outputs (OPi^{new}) as step (6) of algorithm 4.




$$\Delta l_1 = \frac{f^{-1}(p_1)}{\text{Sum of products of neuron } (op_1)} = \frac{f^{-1}(P1)}{C1^{old} * W_{C1,op1}^{old} + C2^{old} * W_{C2,op1}^{old}} \tag{18}$$

By compensating Δl_1 of formula (18) into (17) will get formula (19)


$$OP1^{new} = f[f^{-1}(P1)] = P1 \Rightarrow \text{the 1st Original input_data} \tag{19}$$

Algorithm (4) the Decryption process of the lite-IHNC.

Algorithm (4): Decryption steps of the Instant Hybrid Neuro-Cryptography (IHNC) based on " $\Delta\ell$ " for the proposed Decryptor of the figure (14)

1. Load the same Private key ($W_{C1,OP1}$ to $W_{C1,OP8}$) and ($W_{C2,OP1}$ to $W_{C2,OP8}$) to their corresponding neurons in the Decryptor as initial values of the weights. 
2. Load the Instant key ($\Delta\ell_1: \Delta\ell_8$) to their corresponding neurons in the Decryptor. 
3. Load the received ciphered-data ($C1^{old}$) & ($C2^{old}$) into its corresponding inputs of Decryptor. 

————— Instant Decryption —————

4. Modify all initial values of weights (Private key) by the received instant key ($\Delta\ell_1: \Delta\ell_8$) such as formula (13 & 14)
5. Modify all received Codes ($C1^{old}$) & ($C2^{old}$) by ($\Delta\ell_1: \Delta\ell_8$) to be ($C1^{new}$) & ($C2^{new}$) as formulas (11 & 12)
6. Run Forward propagation for all neurons in the Decryptor to get all original data from ($OP_1: OP_8$). 

4 Case study and results of a deep-IHNC

In last section, we introduced a lite IHNC to simplify our idea. But in this section will propose deeper one. As long as this proposed Deep-IHNC architecture is deeper, the i-codes and i-keys are too complex to crack. Further, its original data is too safe to be predictable.

In this section, we will not mathematically analyze this application. Because it includes similar equations mentioned above albeit in a deeper way. But we will see its behaviors during encrypting a text as well as decryption process with 100% accuracy.

Therefore, we prepared more complex IHNC system depending on asymmetric autoencoder. All parameters in this application are declared with double precision data type (64 bits) to get high accuracy during its computation process.

The proposed Deep-IHNC supports task parallelism by handling 8 characters (via 8 inputs) simultaneously.

Its Encryptor-unit contains seven layers ($L_a, L_b, L_c, L_d, L_e, L_f$ and L_g) as shown in the Fig. 15. For more complexity in our design, the encoder-stage of the Encoder-unit is completely differing than its decoder-stage. The encoder-stage contains five layers (L_a, L_b, L_c, L_d, L_e), while its decoder contains 2 layers (L_f, L_g). Moreover, the encoder-stage has 8 inputs ($p_1 : p_8$) that divided into 2 groups ($p_1 : p_4$) and ($p_5 : p_8$). Each group is fully connected with 2 neuron (L_{a1}, L_{a2}) and (L_{a3}, L_{a4}) respectively through 8 weights for each group.

A different interface for the decoder-stage output is designed to give more asymmetry to the autoencoder. The 8 neurons ($op_1 : op_8$) of the output layer " L_f " are fully connected with the 3 neurons in the layer " L_g " via 24 weights.

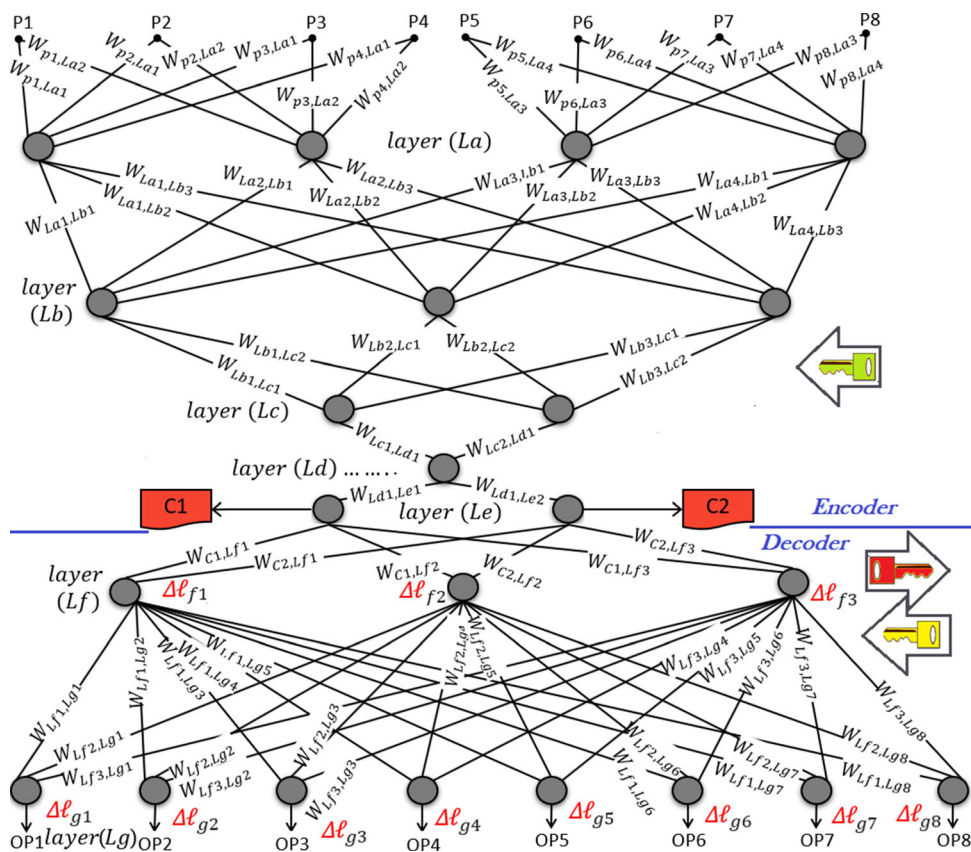
The input-layer " L_a " can accept 8 characters concurrently via its 8 inputs ($p_1 : p_8$). This total number of inputs can be exceeded or reduced depending on the designed system requirements. Each input accepts complete character in ASCII-code format.

We select the ReLU-type as activation function for all neurons in that system, because all inputs have ASCII-codes with positive values greater than one.

According to the architecture of the proposed Deep-IHNC, the encoder-stage has ($4 + 4 + 12 + 6 + 2 + 2 = 30$ weights), so it has public-key with long ($30 * 64 = 1920$ bit). Moreover, its decoder-stage includes ($6 + 24 = 30$) weights. Thus, it has private key with long 1920 bits and i-key with long ($(3 + 8) \Delta\ell_s * 64 = 704$ bits). Its ciphered characters i-code has compressed data ($2 * 64 = 128$ bits). Therefore, the total number of bits that represent 8 characters simultaneously are ($128 + 704 = 832$ bits).

Our scenario here, is encrypting a text of 80 characters as seen in the Fig. 16. The Decryptor is fed forward with sequential sets of characters (eight characters per encryption cycle). Therefore, the FP runs to generate the relevant i-codes as shown in the Fig. 17. The figure shows the changes of 10 i-codes during encryption operations for 10 sets of characters.

Fig. 15 The architecture of the proposed Encryptor of the Deep-IHNC



| | p1 | p2 | p3 | p4 | p5 | p6 | p7 | p8 |
|----|----|----|----|----|----|----|----|----|
| 1 | @ | U | n | i | t | e | d | |
| 2 | N | a | t | i | o | n | s | |
| 3 | m | u | s | t | | e | n | a |
| 4 | c | t | | b | i | n | d | i |
| 5 | n | g | | i | n | t | e | r |
| 6 | n | a | t | i | o | n | a | l |
| 7 | | l | a | w | s | | t | o |
| 8 | | s | t | o | p | | g | r |
| 9 | e | e | n | h | o | u | s | e |
| 10 | | g | a | s | e | s | . | @ |

Fig. 16 Input-text (sets of instant plain data) contains characters

Further, the decoder-stage of the Encryptor is fed backward with the same characters sets (to be their desired targets) to the output layer “ L_g ”. Thus, its BP ML is run to generate ten i-keys based on the previous formulas. Each i-key is represented by eleven learning ratios ($\Delta_a: \Delta_g$) and ($\Delta_{f1}: \Delta_{f3}$) as demonstrated in Fig. 18. This

figure displays sequence changes of 10 samples of i-keys during the BP.

Furthermore, after each encryption cycle, the Encryptor-unit sends its generated i-key (designated for 8 characters) with its associated i-code to the Decryptor-unit.

On other hand, the Decryptor-unit is similar in its architecture to the decoder-stage in the Encryptor. It has 2 layers, input layer “ L_p ” and output layer “ L_q ” with fully connected weights as illustrated in the Fig. 19.

During each decryption cycle, the Decryptor is loaded with the same accepted private-key. It received “i-code” and “i-key”, it performs four main tasks. In the first task, it uses the received i-key to update the received i-code as seen in the Fig. 20. It displays the received 10 samples of old i-codes and their updating (new) i-codes (within the Decryptor-unit) during 10 samples.

In the second task, the received i-key is also used to update all the initial weights (that loaded with private-key) inside the Decryptor as illustrated in the Fig. 21. It shows the old values of all weights (private keys) for all neurons ($q_1 : q_8$) in the layer “ L_q ” along with their updating values during 10 sequential samples of decryptions.

In the third task, the Decryptor runs its FP to restore the original sets of characters. the Fig. 22 shows the recovered original characters across all outputs “ $Out_1 : Out_8$ ” from the eight neurons ($q_1 : q_8$) of layer “ L_q ”.

Fig. 17 The generated i-codes (in our scenario) from the Deep-IHNC

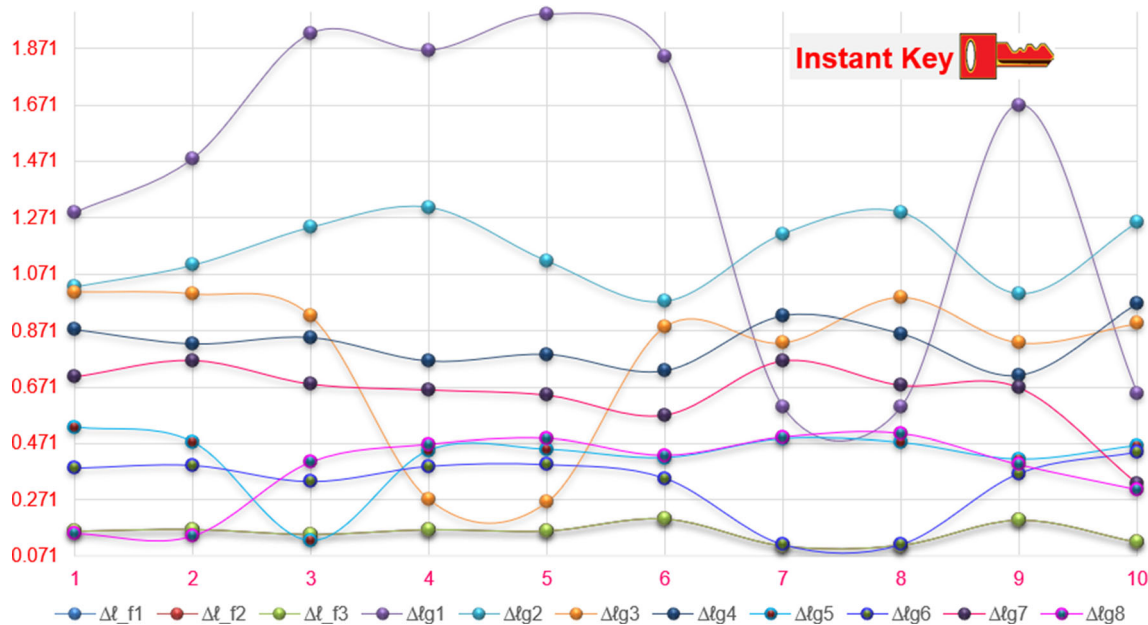
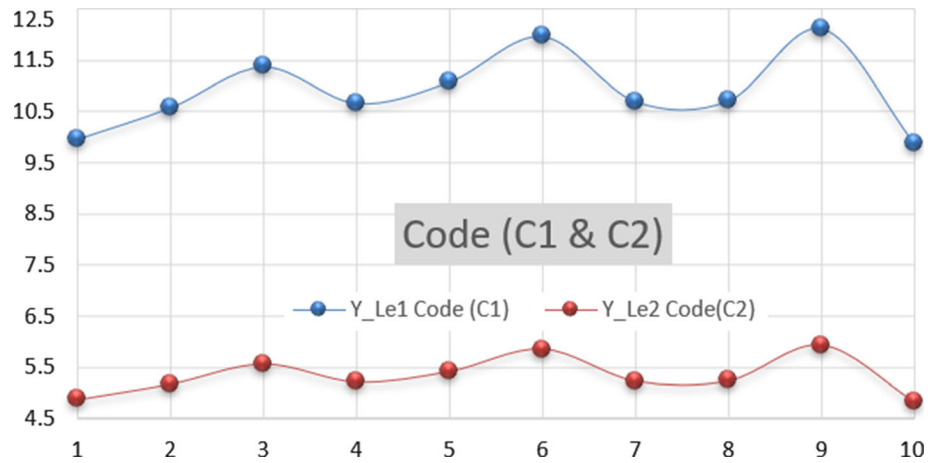


Fig. 18 The Changes of the i-key (10 data samples) in the Deep-IHNC

Eventually, in the fourth task, the Decryptor-unit collects all the character sets retrieved from all the outputs of the layer “ L_q ” in the same sequence (as it encrypted) as shown in the Fig. 23. When we compare the input text to the output text, we find that they are exactly the same.

Eventually, there is a trade-off between building a more in-depth ANN versus its cost and speed of processing. many researchers recommend to implement their systems by using hardware description-language (like VHDL) over the FPGA [22–27].

The mentioned structures of the IHNC systems can be realized using software programs. However, to get more high speed of processing, hardware systems are

recommended. In this case, the designers will face some challenges while writing the IHNC system by VHDL code. We can summarize these challenges in the following points:

- (1) A new datatype (according to Mantissa and Exponent rule) must be added (in VHDL code) to represent all parameters in Float64.
- (2) All IHNC parameters (plain data, weights, learning ratios, outputs...etc.) must be declared by this added Float64 datatype.

Fig. 19 The Decryptor architecture of the Deep-IHNC

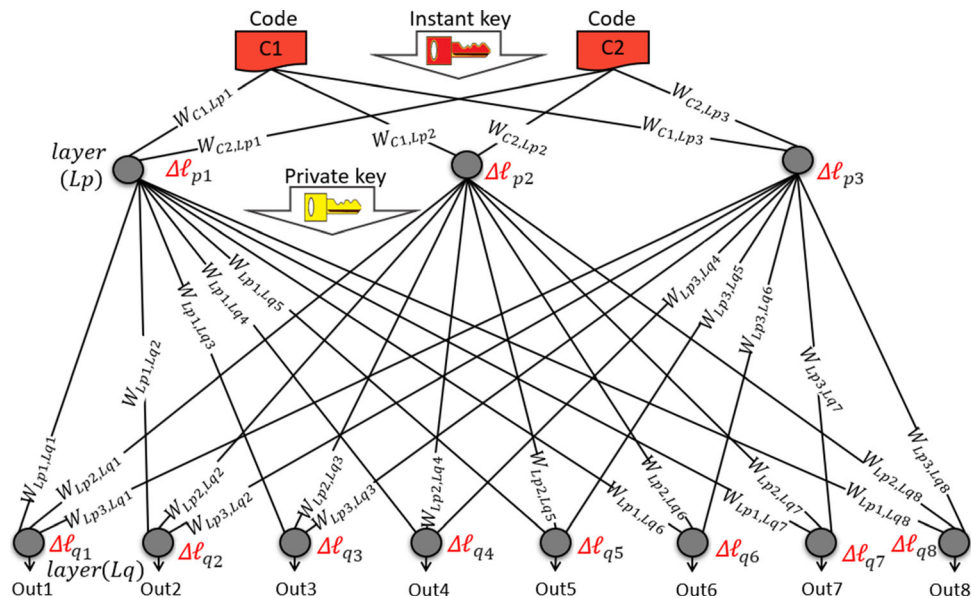
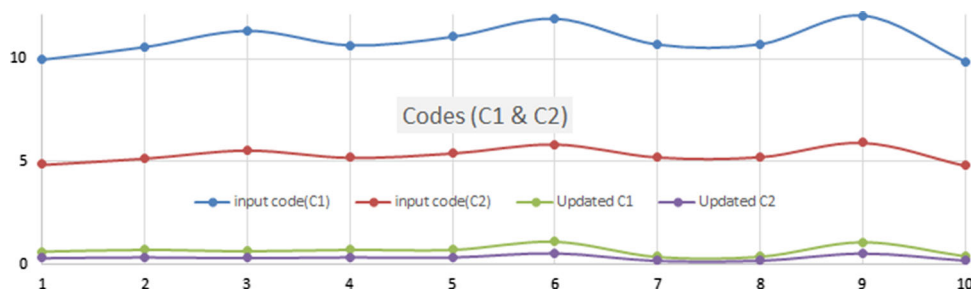


Fig. 20 The updating of the received i-code in of the Deep-IHNC



(3) Three VHDL-functions based on Mantissa and Exponent rule must be written to carry out the following.

- Function1 to make multiplication of Float64 (to perform the products during FP).
- Function2 to make summation of Float64 (to perform the sum of products during FP).
- Function3 to make division of Float64 (to get the learning ratios $\Delta\ell$ during BP).

4.1 Conclusion and hope

In this research, we proposed a new methodology of Hybrid cryptography. It has been realized using the

asymmetric autoencoder based on the new concept of machine learning called the instant learning ratio “ $\Delta\ell$ ”. This Hybrid cryptography “IHNC” serves the task parallelism.

It encrypts multiple data concurrently using Public-key and convert them to compressed instant-code (i-code). Further, it generates instant-key (every encryption process). Moreover, it uses the instant-code as well as the private-key to recover the original data simultaneously.

The proposed hybrid neuro-cryptography does not belong to a specific architecture of neural autoencoder, so each cryptography designer can get creative with her/his design to get their complexity required.



Fig. 21 The Updating of all weights of the layer “L_q” of the Deep-IHNC

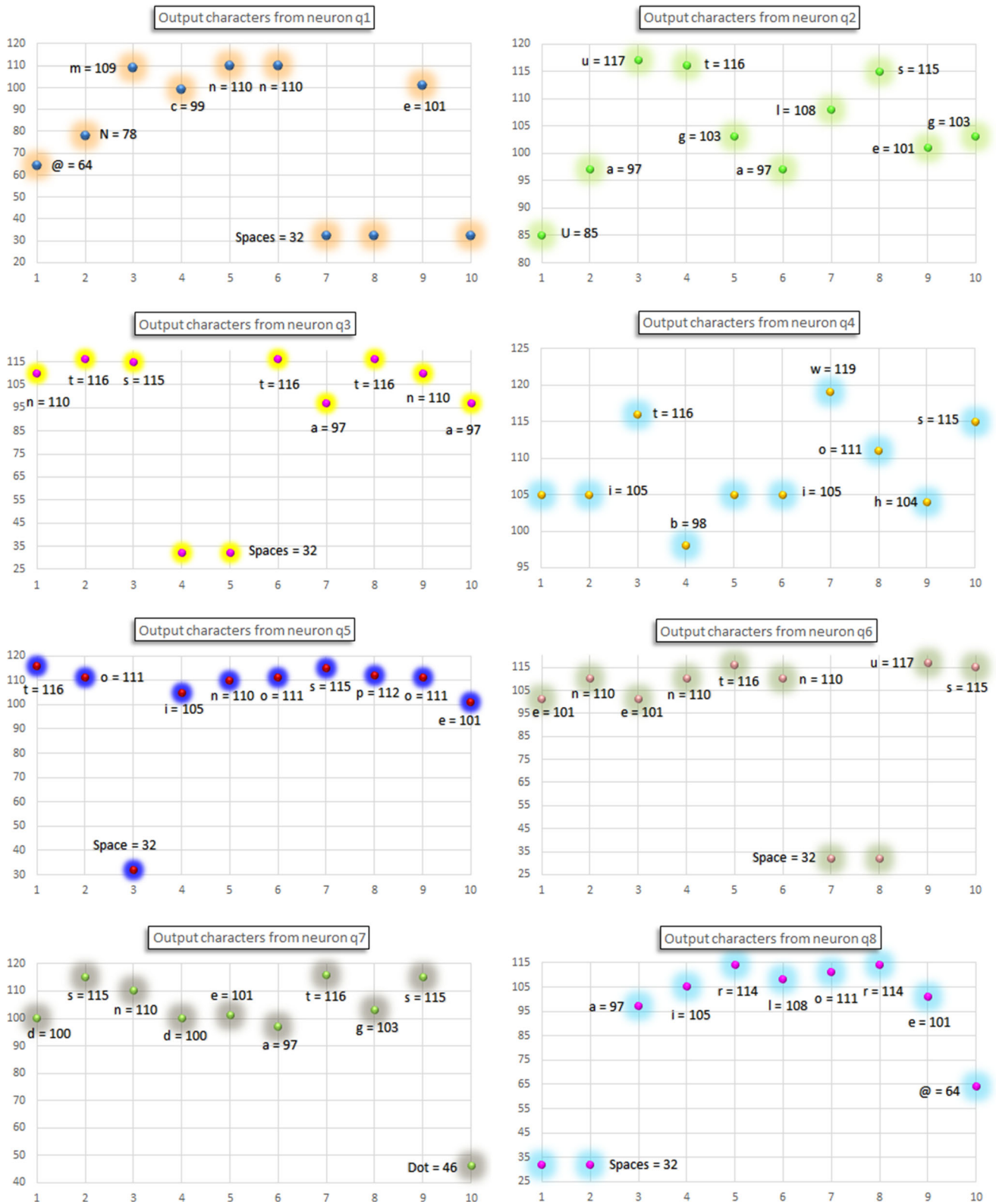


Fig. 22 All recovered characters from layer “ L_q ” in the Deep-IHNC

| | Out1 | Out2 | Out3 | Out4 | Out5 | Out6 | Out7 | Out8 |
|----|------|------|------|------|------|------|------|------|
| 1 | @ | U | n | i | t | e | d | |
| 2 | N | a | t | i | o | n | s | |
| 3 | m | u | s | t | e | n | a | |
| 4 | c | t | | b | i | n | d | i |
| 5 | n | g | | i | n | t | e | r |
| 6 | n | a | t | i | o | n | a | l |
| 7 | | l | a | w | s | | t | o |
| 8 | | s | t | o | p | | g | r |
| 9 | e | e | n | h | o | u | s | e |
| 10 | g | a | s | e | s | . | @ | |

Fig. 23 The recovered original characters on the output-text from the Deep-IHNC

In the future, I hope to design and implement this Hybrid cryptographic system based on the VHDL and FPGA to make the encryption/decryption time within a few clocks for each data block.

Eventually, I hope to use this proposed Hybrid cryptographic rule in an application contains several end-to-end Encryptor/Decryptor units.

Funding Open access funding provided by The Science, Technology & Innovation Funding Authority (STDF) in cooperation with The Egyptian Knowledge Bank (EKB).

Declarations

Conflict of interest The authors whose names are listed immediately below certify that they have NO affiliations with or involvement in any organization or entity with any financial interest (such as honoraria; educational grants; participation in speaker's bureaus; membership, employment, consultancies, stock ownership, or other equity interest' and expert testimony or patent-licensing arrangements), or non-financial interest (such as personal or professional relationships, affiliations, knowledge or beliefs) in the subject matter or materials discussed in this manuscript.

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless

indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit <http://creativecommons.org/licenses/by/4.0/>.

References

- Mohd BJ, Hayajneh T, Yousef KM, Khalaf ZA, Bhuiyan MZ (2018) Hardware design and modeling of lightweight block ciphers for secure communications. *Future Generation Comput Syst* 83:510–521
- Hamed G, Marey M, El-Sayed SA, Tolba MF (2016), Hybrid technique for steganography-based on DNA with n-bits binary coding rule, *IEEE*
- Cassal-Quiroga BB, Campos-Cantón E (2020) Generation of dynamical s-boxes for block ciphers via extended logistic map. *Math Probl Eng.* <https://doi.org/10.1155/2020/2702653>
- Shi J, Chen S, Lu Y, Feng Y, Shi R, Yang Y, Li J (2020) An approach to cryptography based on continuous-variable quantum neural network. *Sci Reports* 10(1):1–3
- Zhang Y, Xue T, Zhai Z, Ma C, Cai X (2008) The Improvement of Public Key Cryptography Based on Chaotic Neural Networks, *IEEE*
- Jhajharia S, Mishra S, Bali S (2013) Public key cryptography using neural networks and genetic algorithms. *IEEE Trans Neural Netw Learn Syst* 31(11):4999–5004
- Dong T, Huang T (2019) Neural cryptography based on complex-valued neural network. *IEEE Trans Neural Netw Learn Syst* 31(11):4999–5004
- Kim I, Park JH, Hwang SO (2020) An efficient public key functional encryption for inner product evaluations. *Neural Comput Appl* 32(17):13117–13128
- Njitacke ZT, Isaac SD, Nestor T, Kengne J (2021) Window of multistability and its control in a simple 3D Hopfield neural network: application to biomedical image encryption. *Neural Comput Appl* 33(12):6733–6752
- Lakshmi C, Thenmozhi K, Rayappan JB, Rajagopalan S, Amirtharajan R, Chidambaram N (2021) Neural-assisted image-dependent encryption scheme for medical image cloud storage. *Neural Comput Appl* 33(12):6671–6684
- Patel S, Thanikaiselvan V, Pelusi D, Nagaraj B, Arunkumar R, Amirtharajan R (2021) Colour image encryption based on customized neural network and DNA encoding. *Neural Comput Appl* 33(21):14533–14550
- Elhoseny M, Shankar K, Lakshmanaprabu SK, Maselena A, Arunkumar N (2020) Hybrid optimization with cryptography encryption for medical image security in Internet of Things. *Neural Comput Appl* 32(15):10979–10993
- Shara J (2020) Some applications of machine learning in cryptography, *Science and Technology Publishing*
- Prabhakaran V, Kulandasamy A (2021) Hybrid semantic deep learning architecture and optimal advanced encryption standard key management scheme for secure cloud storage and intrusion detection. *Neural Comput Appl* 33(21):14459–14479
- Saraswat P, Garg K, Tripathi R, Agarwal A (2019) Encryption algorithm based on neural network, *IEEE*
- Husein AM, Harahap M, Dharmata AM (2019) Hybrid-AES-Blowfish algorithm: key exchange using neural network, *IEEE*
- Badr A (2021) Awesome back-propagation machine learning paradigm. *Neural Comput Appl* 33(20):13225–13249

18. Demertzis K, Iliadis L, Tziritas N, Kikiras P (2020) Anomaly detection via blockchained deep learning smart contracts in industry 4.0. *Neural Comput Appl* 32(23):17361–17378
19. Prabhakaran V, Kulandasamy A (2021) Hybrid semantic deep learning architecture and optimal advanced encryption standard key management scheme for secure cloud storage and intrusion detection. *Neural Comput Appl* 33(21):14459–14479
20. Quinga-Socasi F, Velastegui R, Zhinin-Vera L, Valencia-Ramos R, Ortega-Zamorano F, Chang O (2020) Digital cryptography implementation using neurocomputational model with autoencoder architecture, In: Conference: 12th international conference on agents and artificial intelligence
21. Majumdar A, Tripathi A (2017) Asymmetric stacked autoencoder, IEEE
22. Badr A (2018) Modifying the logic gate symbols to enrich the designing of the computer systems by 3-D bit-matrices. *Ain Shams Eng J* 9(4):3207–3216
23. Badr A (2020) Introducing two complementary novel algebraic operations: matrix-separation and Matrices-joining for programming evaluation and development. *Ain Shams Eng J* 11(2):351–362
24. Badr A, Fouda, A (2013) Design modified architecture for MCS-51 with innovated instructions based on VHDL. *Ain Shams Eng J* 4(4):723–733
25. Badr A (2022) Designing Module to Perform Fast Light Block Cipher (LBC) within microcontrollers by VHDL. *Int J Eng Modern Sci* 1(1):1–9
26. Sun W, Wang J, Zhang N, Yang S (2020) scalable implementation of hippocampal network on digital neuromorphic system towards brain-inspired intelligence. *Appl Sci* 10(8):2857
27. Yang S, Wang J, Zhang N, Deng B, Pang Y, Azghadi MR (2021) CerebelluMorphic: Large-scale neuromorphic model and architecture for supervised motor learning. *IEEE Trans Neural Netw Learn Syst.* <https://doi.org/10.1109/TNNLS.2021.3057070>

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.