Light-weight Internet-of-Things Device Authentication, Encryption and Key Distribution using End-to-End Neural Cryptosystems

Yingnan Sun, *Student Member, IEEE,* Frank P.-W. Lo, *Student Member, IEEE,* and Benny Lo, *Senior Member, IEEE*

***Abstract*—Device authentication, encryption, and key distribution are of vital importance to any Internet-of-Things (IoT) systems, such as the new smart city infrastructures. This is due to the concern that attackers could easily exploit the lack of strong security in IoT devices to gain unauthorized access to the system or to hijack IoT devices to perform denial-of-service attacks on other networks. With the rise of fog and edge computing in IoT systems, increasing numbers of IoT devices have been equipped with computing capabilities to perform data analysis with deep learning technologies. Deep learning on edge devices can be deployed in numerous applications, such as local cardiac arrhythmia detection on a smart sensing patch, but it is rarely applied to device authentication and wireless communication encryption. In this paper, we propose a novel lightweight IoT device authentication, encryption, and key distribution approach using neural cryptosystems and binary latent space. The neural cryptosystems adopt three types of end-to-end encryption schemes: symmetric, public-key, and without keys. A series of experiments were conducted to test the performance and security strength of the proposed neural cryptosystems. The experimental results demonstrate the potential of this novel approach as a promising security and privacy solution for the next-generation of IoT systems.**

***Index Terms*—Deep learning, cryptography, authentication, encryption, Internet-of-Things, binary latent space.**

I. Introduction

DEVICE authentication, encryption, and key distribution are of vital importance to Internet-of-Things (IoT) systems. The implementation of strong security is required for IoT devices to prevent attackers from hijacking the IoT systems to steal personal information and launching denial-of-service attacks on other networks. With the rising popularity of deep learning in security research (for example, in biometrics), many researchers have begun to explore the possibility of using deep learning methods for IoT security applications, such as device authentication and communication encryption. Deep learning has been adopted in many network-level security applications, including intrusion detection, attack modelling, and access control systems. However, it has not been widely exploited in enhancing device-level security in IoT systems.

The concept of neural cryptography was first introduced in 2002 [1], [2]. The black-box properties of neural networks are suitable for cryptographic applications, as the underlying mechanism can be as discrete as possible. Although neural cryptography cannot yet outperform the traditional methods, it is still an emerging research field which has many potential advantages over the traditional cryptographic algorithms used in the next-generation, AI-oriented IoT systems. For example, many deep learning algorithms can be divided into different stages, and one of the trends is to pre-process the data using neural networks on the edge IoT devices to avoid transmitting raw data. It can be more efficient when using channel bandwidth during data streaming, as the processed data is generally less dense and therefore, there will be less information leakage comparing to sending the raw data directly. In this scenario, neural cryptography can be combined with these algorithms, as they can be trained together as one model to further improve the efficiency of wireless transmission.

Y. Sun, F. Lo, and B. Lo are with the Hamlyn Center, Imperial College London, London, UK, e-mail: (y.sun16, po.lo15, benny.lo@imperial.ac.uk
Manuscript received April 19, 2005; revised August 26, 2015.

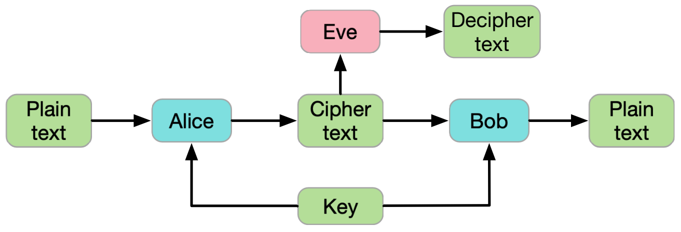


Fig. 1: Symmetric adversarial neural cryptosystem [6]

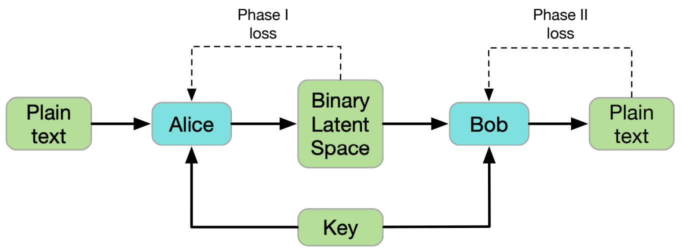
In the existing body of literature, neural cryptography research has been applied primarily to non-IoT systems, as one of the challenges in applying it to these systems is that IoT devices often lack the computational ability to run any deep learning algorithms. However, with the advances in fog and edge computing in IoT systems, many IoT devices, such as Nvidia Jetson Nano [3] and Raspberry Pi [4], are now capable of performing on-node deep learning algorithms. With deep learning frameworks, such as TensorFlow Lite for micro-controllers [5], such devices can execute deep learning algorithms rather than being reliant on centralized servers.

Recently, Abadi and Andersen [6] introduced the concept of adversarial neural cryptography with generative adversarial networks (GANs). A simple GAN model contains a generator and a discriminator. Given a specific training dataset, the generator can learn from the training dataset and generate new data based on the training dataset's statistics. In the intervening time, the discriminator can learn to distinguish between the data generated by the generator and the data in the training dataset. In adversarial neural cryptography, three neural networks are defined, namely Alice, Bob, and Eve, the same as those used in modern cryptography definitions. Alice encrypts plaintext into ciphertext via a secret key that only Alice and Bob possess and sends the ciphertext to Bob, whereas Eve receives the ciphertext and attempts to recover the plaintext without the secret key. This process is illustrated in Fig. 1, where Alice, Bob and Eve are replaced with neural networks in the adversarial neural cryptography. Alice is adversarially trained to make ciphertext more difficult for Eve to recover but possible for Bob to recover with the aid of the secret keys.

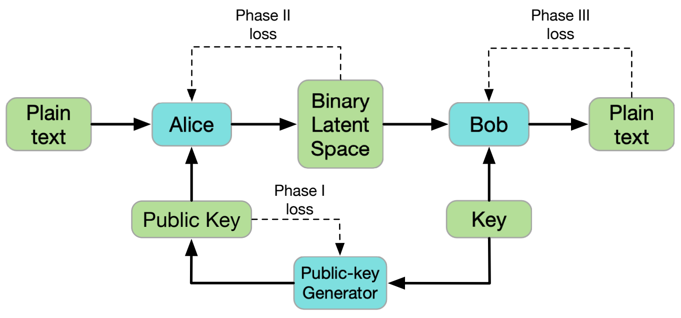
Adversarial neural cryptography offers a new method of training neural networks to perform cryptographic algorithms. However, there are still challenges when applying such neural cryptographic techniques in IoT systems. First, it takes a considerable amount of time to fully train an adversarial neural cryptosystem, and it sometimes fails to converge [6]. Second, the ciphertext produced by GAN and other types of neural networks are formatted as floating point numbers, which are more difficult to analyze for their security strength than binary sequences. Third, the neural networks employed in previous research, such as [6], are too large to be implemented on IoT devices. The ideal neural cryptosystems to be deployed on IoT devices are light-weight, easily, promptly and efficiently trainable at a large scale. For example, neural cryptosystems with only one convolutional layer encryptor can be easily attached onto another neural network which could be used for data pre-processing.

To reduce the complexity and difficulty of training adversarial neural networks, the neural cryptosystem should be trained to let Alice to produce ciphertext that is uncorrelated with the original plaintext and only decipherable by Bob. If Eve is removed, the network model becomes more similar to an auto-encoder, in which the encoder Alice encodes plaintext into latent space (latent vectors), and the decoder Bob decodes the plaintext from the latent space. In a typical auto-encoder model, the latent space represents the plaintext. Latent space is therefore heavily correlated with the plaintext. In contrast, in neural cryptosystems, the encoder must be trained to randomize the latent space to hide the plaintext. Ideally, Alice should behave like a random number generator, producing statistically unpredictable random latent space that secretly contains the plaintext. However, teaching the statistical concepts of neural networks, such as distributions and randomness, has not been widely explored in the literature.

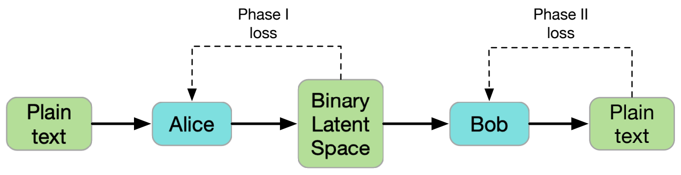
In this paper, we propose a novel light-weight IoT device authentication, encryption, and key distribution approach using end-to-end neural cryptosystems and binary latent space. The training of neural network models is divided into up to three different phases: first, train Alice to produce randomized binary latent space as ciphertext that contains the plaintext. Second, train Bob to decipher the binary latent space into plaintext. In the third phase, which is only required in public-key encryption scheme, the system trains the public key generator networks to produce public keys based on the secret keys generated in the second phase. The proposed cryptosystems can be used to authenticate IoT devices, encrypt sensor data, or distribute encryption keys for further secure communications



(a) End-to-end symmetric neural cryptosystem



(b) End-to-end public key neural cryptosystem



(c) End-to-end neural cryptosystem without keys

Fig. 2: Diagrams of the proposed end-to-end neural cryptosystems: (a) symmetric, (b) public key, and (c) without keys

using other encryption schemes between IoT devices and servers. Our approaches have the potential to be integrated and applied in many deep learning applications to provide a new way of securing an increasing large volume of data generated from the AI-based decision-making algorithms in the new generation of smart city and industrial applications.

The proposed types of end-to-end neural cryptosystem models with different encryption schemes are proposed in this paper, as shown in Fig. 2. The first is a typical symmetric cryptosystem, requiring a pre-shared secret key for both Alice and Bob. The second model is a public key cryptosystem which employs a public key generator neural network to generate public keys. The third model removes the requirement of keys entirely, which is less secure but can be used to send one-time secret keys as plaintext for further encryption. In the adversarial neural cryptosystem in [6], Eve is employed to assess the security strength of the cryptosystems. However, Eve, as another neural network, is often not able to represent real-world attackers, who can employ traditional crypto-analysis tools to retrieve secret keys. Therefore, a more traditional crypto-analysis is adopted in this paper to assess the security strength of the proposed neural cryptosystems, rather than analyzing how adversarial neural networks (Eve) can attack cryptosystems.

TABLE I: Summary of related works on neural cryptosystems and deep learning based random number generators

Ref.	Encryption scheme	Data type	Approach
[7]	Symmetric	Text	Send one neural network parameters to the other network via a pre-established secure channel
[8]	Symmetric	Text	Genetic algorithm and error back propagation neural network
[9]	Symmetric	Text	Symmetric cryptosystems based on back propagation of neural networks
[10]	Symmetric	Key	Symmetric cryptosystems with neural networks synchronized by tree parity machine
[11]	Symmetric	Images	Weights transmission between two copies of echo state networks as encryptor and decryptor
[12]	Symmetric	Text	Radial basis function networks with natural noise simulations
[13]	Symmetric	Text	Adversarial training of spectrum-diverse unified neuroevolution neural networks
[14]	Symmetric	Text	Multiple auto-encoder networks as secret key generators and hash value generators separately
[15]	Symmetric	Text	Symmetric adversarial neural cryptosystems with multiple attackers (Eve) having partial keys
[16]	Symmetric	Text	Symmetric adversarial neural cryptosystems with a chosen-plaintext attack based attacker (Eve)
[17]	Public key	Text	Public key cryptosystems with identical neural networks initialized by two PRNGs with shared one-time seeds

The rest of the paper is organized as follows. Related works on deep learning-based cryptosystems are reviewed in section II. Section III presents the detailed methodology of the proposed end-to-end neural cryptosystems with three types of encryption schemes. The results, evaluations, and security analysis of the proposed neural cryptosystems are presented in section IV, follow by conclusions and future research directions in section V.

II. Literature Review

In the past, deep learning has not been widely used in cryptographic applications for IoT systems due to the difficulty of implementing the highly complex algorithms on IoT devices with very limited computational power. Therefore, most neural cryptosystems research has been focused on data protection for computer networks. In addition, the majority of neural cryptosystems in the literature only add or replace parts of cryptographic algorithms with neural networks, rather than end-to-end encryption schemes with only neural networks, as proposed in this paper.

One of the first approaches proposed in neural cryptography was to use two neural networks with identical parameters as a cryptosystem for symmetric encryption schemes. Yayik and Kutlu [7] proposed a symmetric neural cryptosystem in which the neural network parameters of the encryptor, such as weights, number of neurons, and number of hidden layers, were securely transmitted to the decryptor via a pre-established secured channel. The output of a neural network can also be reversed by back propagation of any neural networks with the same parameters as demonstrated in [8] and [9]. Neural cryptosystems can also solve the issue of the distribution of secret keys for symmetric encryption schemes [10], as two synchronized neural networks can exchange secret keys securely in a public channel. Synchronization between the encryptor and decryptor can also be achieved by transmitting partial weights of the neural networks so that the adversarial neural network cannot use the weights directly, as in [11]. While using two neural networks for symmetric encryption has been widely explored in the literature, it requires the neural network architectures that either have inverse operations or a part of the neural network parameters can be transmitted over the communication channel.

Neural networks with chaotic properties can also be used in neural cryptosystems, which introduces randomness into the neural networks themselves. For example, [12] proposed the use of radial basis function networks with natural noise simulations as one-to-one personal encryption algorithms to protect user privacy within cloud servers. Other network architecture can also be used in neural cryptosystems, such as topology evolving neural networks [13] and multi-layer auto-encoder neural networks [14]. However, the majority of the aforementioned neural cryptosystems are not end-to-end encryption schemes, as at least one part of the cryptographic functionality is not performed by the neural networks.

The adversarial neural cryptography proposed in [6] is a promising end-to-end neural cryptography approach. As previously mentioned, adversarial training introduces Eve as another neural network in the cryptosystem and as one that is frequently trained with less information than is provided to Bob. In [6], Eve only receives the ciphertext, whereas Bob has both the secret keys and the ciphertext. A small advantage is given to Eve by increasing Eve's number of training iterations compared to both Alice and Bob. Alice needs to make ciphertext more difficult for Eve to decipher, therefore enhancing the security level of the cryptosystems. More advantages can be given to Eve to increase the security level of the cryptosystems, for example, in [15], two symmetric neural cryptosystem approaches, based on adversarial neural networks, were proposed. The first approach assumes that the attacker neural network (Eve) obtains a part of the secret keys and the ciphertext, to decipher plaintext or output complete secret keys. The second approach employs two attacker neural networks (two Eves that know either partial secret keys or the ciphertext) to predict secret keys or decipher the plaintext.

Traditional cryptographic attacks can also be utilized in neural cryptography to increase the security strength of the neural cryptosystems. For example, chosen-plaintext attacks are used in the adversarial neural cryptosystem proposed in [16] where Eve produces two plaintexts individually for Alice to send and distinguishes which of the two is encrypted

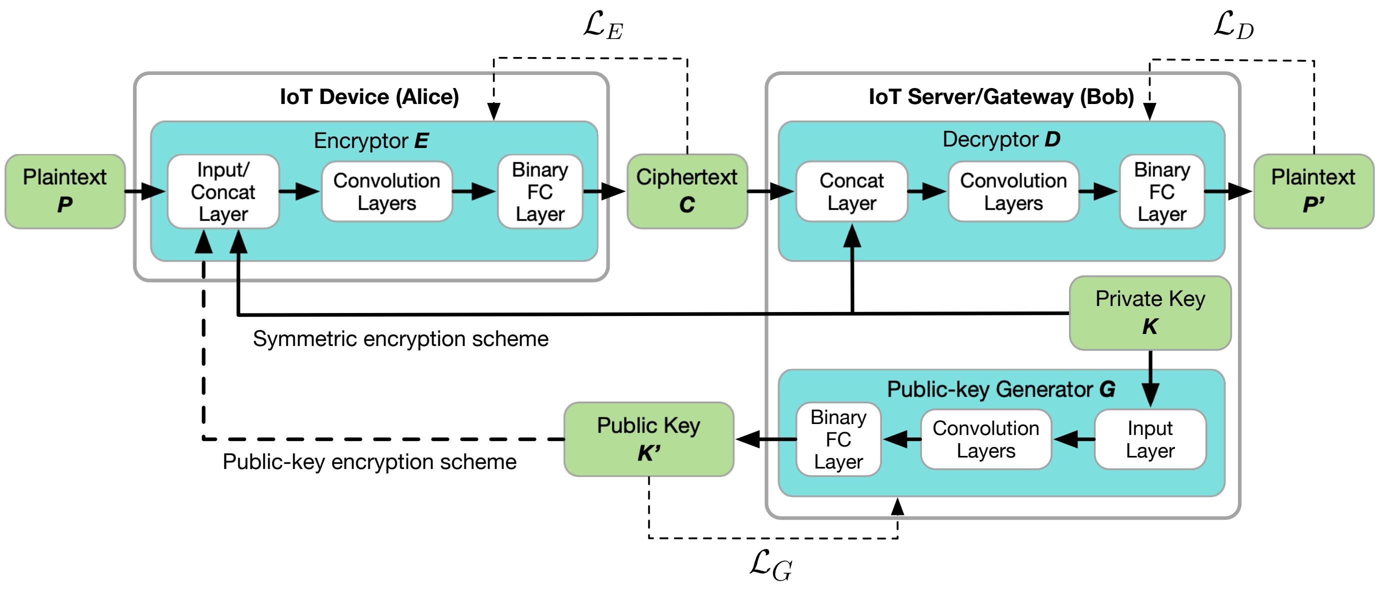


Fig. 3: Illustration of the proposed light-weight neural cryptosystems

in the current ciphertext. In this scenario, Alice must be cleverer in how it hides the plaintext within the ciphertext. A major weakness in all of the aforementioned adversarial neural cryptosystems is that due to the nature of the adversarial training, not all training trials can reach the desired condition, with the result that Eve can neither decipher the plaintext nor obtain secret keys. The success rate of adversarial training heavily depends on the amount of information given to Eve and the complexity of the chosen neural network architecture.

There are several advantages of neural cryptosystems proposed in this paper over the state-of-the-art neural cryptosystems. First, the proposed neural cryptosystems are trained using a phased training approach rather than the adversarial training approach adopted in many adversarial neural cryptosystems. For example, for the proposed symmetric neural cryptosystems, Alice is trained in the first phase, whereas Bob is trained in the second phase. This successive and phased training ensures that the neural network models will always reach the desired conditions, unlike in the adversarial training approaches. Second, in the proposed neural cryptosystems, no redundant neural network parameters is required to be transmitted during or before the authentication and key distribution process, greatly reducing the overhead in the wireless communication channels. This is also important for most of the IoT systems where a large number of IoT devices are likely to communicate simultaneously with the servers. Third, instead of using two copies of the same neural networks on Alice and Bob, the proposed cryptosystems shift most of the computational complexity of the neural networks to Bob, who is usually the IoT server with considerably more computational resources. Finally, the proposed cryptosystems output binary numbers as the ciphertext rather than the floating numbers used in most adversarial neural cryptosystems. This allows traditional crypto-analysis, such as randomness tests, to be used for testing the strength of the security of neural cryptosystems. A thorough security analysis of the ciphertext produced by the proposed cryptosystems was conducted to demonstrate the strength of the security of the proposed cryptosystems and it is shown in the results section.

III. Methodology

In this section, the proposed end-to-end neural cryptosystems for device authentication, encryption and key distribution of IoT-based system security will be explained in detail. Simplified diagrams of the three types of the encryption schemes are shown in Fig. 2, and these are described using the same method as in classical cryptographic scenarios. A more detailed illustration of all three types of the encryption schemes of the proposed neural cryptosystem are presented in Fig. 3. Solid arrow lines represent the flow of private keys and dashed arrow lines represent the flow of public keys. The dashed arrow lines are only used in the public-key encryption scheme. A more detailed individual explanation of the implementation of each encryption scheme is provided, alongside the details of the implementation of device authentication and key distribution process.

A. Symmetric Neural Cryptosystem

In all the proposed cryptosystems, Alice represents an IoT device and Bob represents an IoT server or gateway. For symmetric encryption scheme of the proposed cryptosystem, as shown in Fig. 3, Bob uses a decryptor D and a N-bit private key K, which is generated by a random number generator that is not shown in Fig. 3. On the other side, Alice encrypts the plaintext P into ciphertext C via an encryptor E and a pre-distributed K. Assuming the parameters of the neural network in E is θ_E, C can be expressed as

$$C = E(\theta_E, P, K) \tag{1}$$

where E consists of an input layer, convolution layers, and a binary Fully-Connected (FC) layer. Two binary inputs, the plaintext P and K are concatenated at the input layer, then fed into the convolution layers. The output of the final convolution layer is fed into a FC layer with a binary sigmoid activation function, which outputs $\{0, 1\}$. As C is a randomized binary space that contains the plaintext, the distribution of 0s and 1s should be uniform. Therefore, a novel loss function $\mathcal{L}_E$ is proposed to train the neural network to randomize its outputs. To avoid the neurons of the binary FC layer output same 0s or 1s for a succession of plaintext inputs, $\mathcal{L}_E$ considers distribution of 0s and 1s over an entire mini-batch. Assume the batch size is B and the number of neurons of the binary FC layer is M, the size of the outputs of the encryptor (denoted as $\mathbb{C}$) over the entire mini-batch is $B \times M$. Assuming the output of each neuron of the binary FC layer is c, $\mathbb{C}$ can be expressed as

$$\mathbb{C} = \begin{bmatrix} c_{1,1} & c_{1,2} & \cdots & c_{1,M} \\ c_{2,1} & c_{2,2} & \cdots & c_{2,M} \\ \vdots & \vdots & \ddots & \vdots \\ c_{B,1} & c_{B,2} & \cdots & c_{B,M} \end{bmatrix} \tag{2}$$

where $c_{b,m}$ is the output of the m^{th} neuron of the b^{th} sample in a mini-batch.

Statistically, if B and M are significantly large, the appearances of 0s and 1s in $\mathbb{C}$ should be equal. To train E, the loss function is divided into two parts. The mean of the outputs of E over the b^{th} sample should be close to 0.5, therefore, the first part of the loss function, $\mathcal{L}_{E_b}$, can be expressed as

$$\mathcal{L}_{E_b} = \frac{1}{M}\sum_{m=1}^{M} c_{b,m} - 0.5 \tag{3}$$

in the mean time, the mean of the outputs of the m^{th} neuron over the entire mini-batch should also be close to 0.5. This forces the neurons to unpredictably output 0s and 1s, increasing the randomness of $\mathbb{C}$. Similarly, the second part of the loss function $\mathcal{L}_{E_m}$ can be written as

$$\mathcal{L}_{E_m} = \frac{1}{B}\sum_{b=1}^{B} c_{b,m} - 0.5 \tag{4}$$

therefore, the final loss function for the encryptor can be expressed as

$$\mathcal{L}_E = \frac{1}{B}\sum_{b=1}^{B} |\mathcal{L}_{E_b}| + \frac{1}{M}\sum_{m=1}^{M} |\mathcal{L}_{E_m}| \tag{5}$$

where $\mathcal{L}_{E_b}$ and $\mathcal{L}_{E_m}$ are averaged across the entire mini-batch and the entire sample respectively.

By minimizing $\mathcal{L}_E$, the encryptor is able to output randomized binary ciphertext. Although the encryptor is not taught to hide the P specifically, the plaintext has been hidden in the ciphertext, which will be verified in the results section. On the IoT server (Bob) side, the output of the decryptor D is the deciphered plaintext, denoted as P', which can be written as

$$P' = D(\theta_D, C, K) \tag{6}$$

where θ_D is the parameters of the neural network of the decryptor. The loss function for optimizing D is $\mathcal{L}_D$ as shown

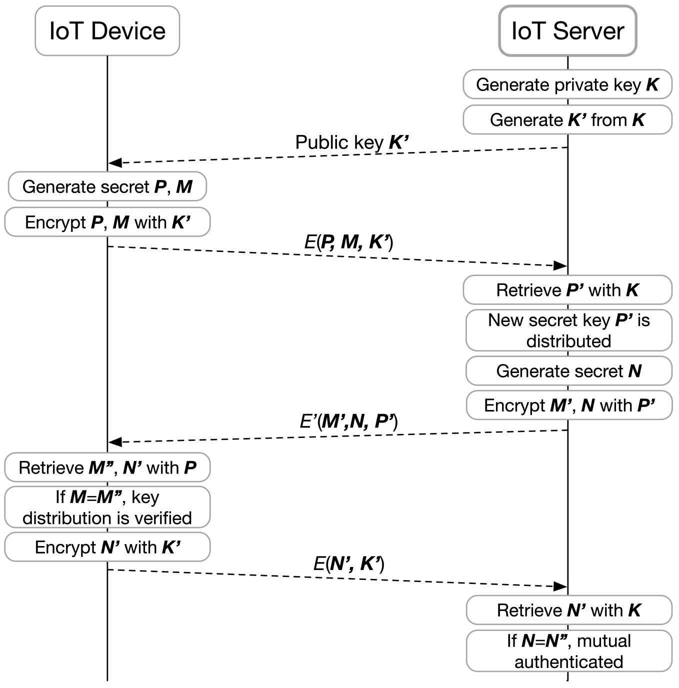


Fig. 4: Illustration of the key distribution and authentication process with the proposed public-key neural crytosystem

in Fig. 3. $\mathcal{L}_D$ is the L1 distance between P and P', which can be expressed as

$$\mathcal{L}_D = \frac{1}{N}\sum_{n=1}^{N} |P' - P| \tag{7}$$

B. Public-key Neural Cryptosystem

For public-key encryption scheme of the proposed neural cryptosystem, Bob consists of the decryptor D and a public-key generator G, which is also a neural network. The private key K is fed into G to generate a public key K' and sent to Alice via a public channel. The training of the public-key generator follows the same principle applied to the encryptor. The mathematical expression for G and the loss function $\mathcal{L}_G$ are identical to E and $\mathcal{L}_E$ respectively. After receiving K', Alice encrypts plaintext P into ciphertext C via the same encryptor E, which has identical mathematical expression as that of the symmetric neural cryptosystem.

Fig. 4 shows a detailed illustration of the key distribution and mutual authentication process between the IoT device and the IoT server with the proposed public-key cryptosystem. A similar process can be followed for the other two proposed cryptosystems. When the IoT device receives the public key K' from the IoT server, it encrypts a secret key P and a secret message M with K'. The encrypted ciphertext $C = E(P, M, K')$ is then sent to the IoT server and decrypted with the private key K subsequently. The network parameter θ is omitted here. The decrypted secret key P' is now distributed to the IoT server from the IoT device. To verify the key distribution and achieve mutual authentication, the IoT server encrypts a newly generated secret message N

and the decrypted secret message M' with P' and sends them to the IoT device. If the IoT device receives and decrypts M'', and it checks that M'' is identical to the original message M, the key distribution is successfully verified. The IoT device then encrypts and sends the decrypted message N' back to the IoT server. If the retrieved N'' is identical to N, the IoT device and the IoT server achieve mutual authentication.

C. Neural Cryptosystem without Keys

The proposed neural cryptosystem can work without any public and private keys. In this scenario, the plaintext P is encrypted into the ciphertext C by the encryptor E without the assistance of the private key K or the public key K', and the decryptor extracts P from C directly. This light-weight approach is vulnerable to chosen plaintext attack; therefore, it is only suitable to send one-time encryption keys. For instance, it can be used to pre-distribute the secret key K for the proposed symmetric neural cryptosystem, or to be used as E' for encrypting M' and N on the IoT server during the verification phase of the key distribution process.

D. Binary Sigmoid Activation

The last neural network layer for E, D, and G is a FC layer with binary sigmoid activation. Assuming the output of the linear operations of the neurons of the FC layer is x, a stochastic binarization function [18] is applied as

$$y = \begin{cases} 1 & \text{with probability } p = \sigma(x), \\ 0 & \text{with probability } 1-p, \end{cases} \tag{8}$$

where y is the output of the activation function and σ is a hard sigmoid function written as

$$\sigma(x) = \text{clip}(\frac{x+1}{2}, 0, 1) = \max(0, \min(\frac{x+1}{2}, 1)) \tag{9}$$

E. Experiments

A series of experiments were conducted to analyze the performance and security strength of the proposed neural cryptosystems. As aforementioned, the private keys and plaintext are random binary sequences generated on the IoT server and devices. As there are many repetitive experiments for investigating the effects of changing the size of the binary latent space, a fixed training dataset and a fixed testing dataset consists of private keys and plaintext are generated. Both the private keys and plaintext have a length of $N = 16$ bits, and the batch size B for training was set to 4,096 for all the experiments, with the exception of the training for the NIST test, where the batch size was set to 256,000.

For the encryptor neural network, the private key and plaintext are concatenated at the input layer, therefore, the input size for the first convolution layer is $2N = 32 \times 1$. The sizes of the convolution layers for the encryptor used in the experiments are listed in Table II, which has the format of $2N$ times the number of filters for each convolution layer. All the convolution layers are one-dimensional and have kernel size of 1, Rectified Linear Unit (ReLU) activation, and 'same' padding scheme. The output of the last convolution layer is flattened

TABLE II: Network parameters of the encryptor for symmetric, public-key, and without key encryption schemes

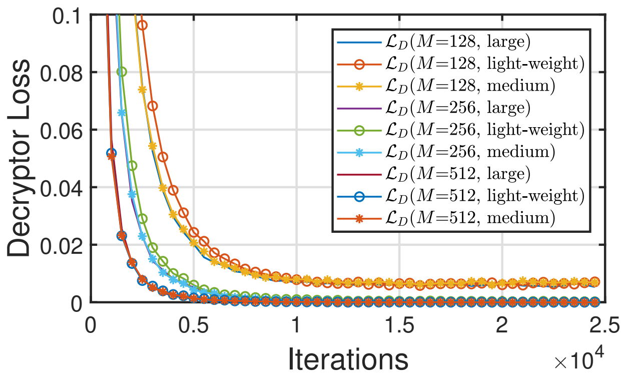
Scheme	Size	Conv 1	Conv 2	Conv 3
Symmetric	Large	32×32	32×32	32×32
	Medium	32×16	32×16	32×16
	Light-weight	32×16	-	-
Public-key	Large	32×32	32×32	32×32
	Medium	32×16	32×16	32×16
Without key	Medium	32×16	32×16	32×16
	Light-weight	32×16	-	-

and fed into the binary FC layer with the number of neurons $M = [128, 256, 512]$. The length of the binary latent space (ciphertext) is equal to the number of neurons in the FC layer. With a larger M, the more communication overhead will be generated, and it would be easier for the decryptor to decipher the ciphertext. The decipher takes both the ciphertext and the private keys as inputs, then concatenate them together. The input to the first convolution layer has a size of $(M + N) \times 1$. The decryptor consists of three convolution layers, which have the same parameters: kernel size of 1, number of filter of 128, ReLU activation, and 'same' padding scheme. The convolution layers of the public-key generator are identical to those of the decryptor. The computational complexity of the proposed neural cryptosystems is mostly on the IoT server, as it does not have the power and resource constrains as per the IoT devices.

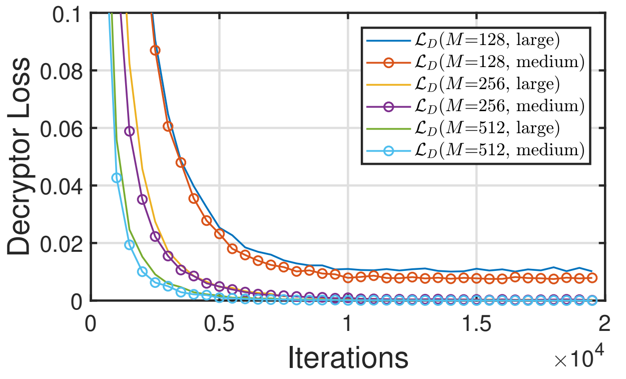
A phased training approach is adopted in the proposed neural cryptosystems, which means the encryptor, decryptor, and public-key generator are trained in a sequential order for all three encryption schemes. While one neural network is being trained, the network parameters of the other two neural networks are frozen. For symmetric neural cryptosystems and cryptosystems without keys, the encryptor is trained first for 10 epochs using Adam optimization [19], then the decryptor is trained for 10 epochs. A staircase exponential decay learning rate was used for all the training with an initial learning rate of 10^{-6}, a decay step of 3,000, and a decay rate of 0.95. For the public-key neural cryptosystem, the order for training is the public-key generator, encryptor, and decryptor, each of which is trained for 10 epochs. The experiments were implemented using Tensorflow 2.0.0 [20] on a Ubuntu 16.04 workstation with a 3.30GHz i9-9820X CPU and four GeForce RTX 2080 GPUs.

IV. Results

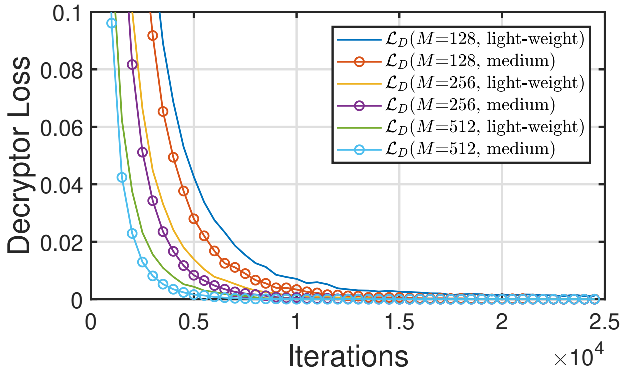
In this section, the results on the performance of the proposed neural cryptosystems, in terms of plain text reconstruction error rates and training and testing accuracy, are presented. In addition, the security strength of the proposed neural cryptosystems is also evaluated using crypto-analysis, since the ciphertext should have high entropy and random to attackers. As phased training was conducted instead of adversarial training, all the models can reach the desired conditions where the training losses for the encryptor, decryptor, and public-key generator are minimized.



(a) Symmetric encryption scheme



(b) Public-key encryption scheme



(c) Encryption scheme without keys

Fig. 5: Decryptor losses of the proposed neural cryptosystems during training

A. Encryptor, Decryptor, and Public-key Generator Loss and Plaintext Reconstruction Error

The three graphs shown in Fig. 5 are the first 2.0 or 2.5×10^4 iterations of the decryptor training losses of the proposed neural cryptosystems, in regards to the type of the encryption schemes, the size of the convolution layers, and the length of the ciphertext. As shown in Fig. 5, the rate of the decryptor loss of a model to reach the minima mostly depends on the length of the ciphertext (M). When M is sufficiently large, the hidden plaintext can be retrieved by the decryptor with very small plaintext reconstruction errors. This is also proven in Table. III, which lists the plaintext reconstruction errors for the decryptors of the proposed neural cryptosystems with $M = [128, 256, 512]$ on the testing dataset.

TABLE III: Plaintext reconstruction errors for the proposed neural cryptosystems with M=$[128, 256, 512]$ on the testing dataset

Scheme	Size	M=128 ($\times 10^{-4}$)	M=256 ($\times 10^{-4}$)	M=512 ($\times 10^{-6}$)
Symmetric	Large	64.2516±3.2130	1.1983±0.4145	3.1433±7.8163
	Medium	71.7176±3.1877	1.1356±0.4164	3.7002±7.5016
	Light-weight	73.6855±3.1877	1.4771±0.4646	2.7236±6.5388
Public-key	Large	123.8960±4.3110	1.7292±0.5222	3.0441±6.9875
	Medium	93.2974±3.7528	1.9272±0.5506	2.4032±6.0401
Without key	Medium	0.1620±0.1640	0.1117±0.1314	25.0551±20.3223
	Light-weight	7.3560±1.0950	0.4010±0.2521	33.1800±22.5048

TABLE IV: Encryptor loss for the proposed neural cryptosystems with M=$[128, 256, 512]$ on the testing dataset

Scheme	Size	M=128 ($\times 10^{-4}$)	M=256 ($\times 10^{-4}$)	M=512 ($\times 10^{-6}$)
Symmetric	Large	0.0410±0.0007	0.0211±0.0006	0.0167±0.0006
	Medium	0.0294±0.0007	0.0214±0.0006	0.0167±0.0006
	Light-weight	0.0270±0.0007	0.0209±0.0006	0.0168±0.0006
Public-key	Large	0.0270±0.0008	0.0219±0.0006	0.0164±0.0006
	Medium	0.0271±0.0007	0.0211±0.0007	0.0167±0.0006
Without key	Medium	0.0267±0.0009	0.0207±0.0008	0.0167±0.0008
	Light-weight	0.0266±0.0009	0.0207±0.0008	0.0165±0.0008

The neural cryptosystem models with medium-size convolution layers for the encryptors perform better than the large-size and light-weight models. As aforementioned, the size of the decryptors for all the models are identical. When there is only one convolution layer in the encryptor, the proposed symmetric and without key neural cryptosystems can still work with only minor decrease in the plaintext reconstruction accuracy. For the public-key neural cryptosystems, only large and medium-size models were trained and the model with medium-size convolution layers performs sufficiently better than the models with large-size convolution layers. When $M = 512$, there is on average 3 bits reconstruction error per million bits of the plaintext transmission, which is acceptable for encryption applications. When $M = 256$, the reconstruction error rises to around 2 bits per 10,000 bits of the plaintext transmission, which is also acceptable for communication encryption but may require the assistant of error correction coding (ECC) schemes, such as the Bose-Chaudhuri-Hocquenghem (BCH) codes [21]. However, when $M = 128$, the reconstruction errors for symmetric, public-key, and without keys are around 7, 10, and 0.7 bits per 1,000 bits of plaintext transmission. The error rates are too large for the neural cryptosystems to be used for encryption applications directly. The ECC schemes can still be applied in this case but with less transmission efficiency.

Table. IV lists the encryptor losses for the three types of neural cryptosystems on the testing dataset. The type of encryption scheme and the size of the convolution layers have less effect on the encryptor loss. When the length of the ciphertext increases, the encryptor loss decreases. Table. V

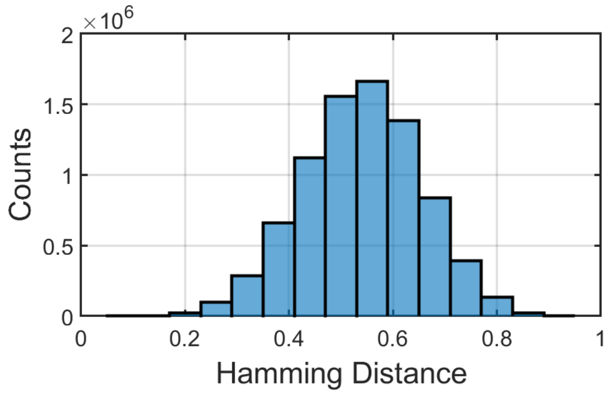


Fig. 6: Distribution of Hamming distances between private keys and the corresponding public keys of the public-key neural cryptosystem when $M = 512$ on the test dataset

TABLE V: Public-key generator loss for the public-key encryption scheme of the proposed neural cryptosystems on the testing dataset

Scheme	Size	M=128 ($\times10^{-4}$)	M=256 ($\times10^{-4}$)	M=512 ($\times10^{-6}$)
Public-key	Large	0.0609±0.0016	0.0602±0.0016	0.0602±0.0016
	Medium	0.0612±0.0016	0.0611±0.0016	0.0606±0.0016

presents the public-key generator losses for the public-key neural cryptosystems on the testing dataset. As the length of the public keys is fixed at N, the public-key generator loss is affected by the size of the convolution layers and the length of the ciphertext.

B. Hamming Distance between Private Key and Public Key

In the public-key neural cryptosystems, any public key generated using the public-key generator, should differ from its pairing private key. Therefore, Hamming distances between pairs of the public key and private key are calculated, and the distribution of the hamming distances are shown in Fig. 6. By definition, the Hamming distance between two binary sequences of the same length, is calculated as the bitwise differences between them [22]. The Hamming distance has a range of $[0, 1]$, and 0 indicates the two binary sequences are identical on every bit position, whereas 1 means every bit position is different. For two binary sequences, complete different means that they can be treated as identical. Therefore, for a large number of binary sequence pairs, the distribution of the hamming distances between these pairs should be a Gaussian distribution whose mean is about 0.5. As shown in Fig. 6, the distribution follows a Gaussian distribution with a mean slightly off 0.5, which indicates the private key cannot be obtained from the corresponding public key.

C. Hamming Distance between Plaintext and Ciphertext

The distribution of Hamming distances between the plaintext and its corresponding ciphertext for the proposed symmetric neural cryptosystem is shown in Fig. 7. As the plaintext has a shorter length than its ciphertext, a sliding window with

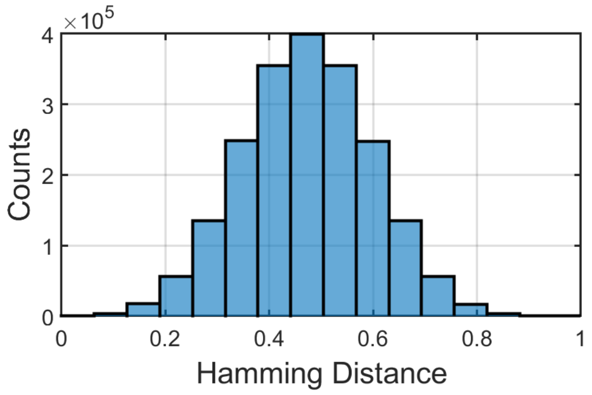


Fig. 7: Distribution of Hamming distances between the plaintext and its ciphertext of the symmetric neural cryptosystem when $M = 512$ and a sliding window with window size of N on the test dataset

a window size identical to the plaintext is adopted. A Gaussian distribution of hamming distances indicates that the plaintext is well hidden in the ciphertext, as it cannot be obtained directly. However, no correlations between the plaintext and its ciphertext cannot prove that the plaintext has been securely hidden. A thorough crypto-analysis is still required to evaluate the security strength of the proposed neural cryptosystems.

D. Security Analysis for Ciphertext

As ciphertext is often transmitted via a public communication channel, it is of vital importance for the neural cryptosystems to generate ciphertext that is random to third-parties without the decryptor and private keys. Therefore, the ciphertext generated on the testing dataset from different models was evaluated using ENT test, a pseudorandom number sequence test suite consists of five individual tests, namely entropy, chi-square, arithmetic mean, Monte-Carlo Pi, and serial correlation tests. These tests were used in [17], [23], [24], for testing the security strength of the cryptosystems.

TABLE VI: Results of the ENT, a pseudorandom number sequence test, for the ciphertext of the proposed neural cryptosystems with $M = 512$ (MC-Pi=Monte-Carlo Pi, Serial Corr.=serial correlation)

Scheme	Size	entropy	chi-square	mean	MC-Pi	Serial Corr.
Symmetric	Large	7.9936	232625.4929	127.4747	3.1517	-0.000053
	Medium	7.9963	133832.0336	127.5103	3.1567	-0.000149
	Light-weight	7.9966	122781.9186	127.4939	3.1571	0.000248
Public-key	Large	7.9908	334673.1258	127.5057	3.1164	0.000527
	Medium	7.9939	223544.8776	127.4908	3.1646	0.000017
Without key	Medium	7.9943	205531.4516	127.5081	3.1476	0.000281
	Light-weight	7.9862	507359.3769	127.4984	3.1639	0.000125
Optimal	-	8	-	127.5	3.1416	0

The results are concluded in Table VI. For the entropy test, entropy represents the information density of the ciphertext, expressed as a number of bits per byte. Therefore, the optimal

TABLE VII: NIST Statistical Test Results for the ciphertext of the proposed neural cryptosystems with $M = 512$ (the minimum pass rate for each statistical test is approximately = 90% for a sample size = 10 binary sequences)

Statistical Tests	P-value	Proportion	Pass/Fail
Frequency	0.534146	100%	Pass
Block Frequency	0.002043	100%	Pass
Cumulative Sums$^+$ (2)	0.630949	100%	Pass
Runs	0.739918	100%	Pass
Longest Run	0.739918	90%	Pass
Rank	0.350485	100%	Pass
FFT	0.035174	100%	Pass
Non Overlapping Template$^+$ (148)	0.460586	97%	Pass
Overlapping Template	0.213309	100%	Pass
Serial$^+$ (2)	0.545202	100%	Pass
Linear Complexity	0.122325	90%	Pass

value for an entropy test should be 8 bits per byte. As presented in the table, the entropy values for all the models are very close to 8, indicating high randomness in ciphertext. The chi-square test is the another commonly used test for assessing the relationships between different variables. The values in the chi-square column is the chi-square distribution of 26,214,400 samples, and randomly it would exceed this value less than 0.01 percent of the times for all models. This indicates that the ciphertext samples are not totally independent from one another. This is acceptable as long as the relationships between two samples cannot be derived easily. Arithmetic mean is the sum of all bytes in a binary sequence and divided by the total number of bytes in that binary sequence. If the arithmetic mean for a binary sequence is close to 127.5, the binary sequence can be considered as random. Monte-Carlo Pi test uses each successive six bytes in a binary sequence to form a 24-bit Cartesian coordinates (X, Y) within a unit square. The estimation of Pi is calculated as the number of the coordinates in the inscribed circle of that square multiplied by four, then divided by the total number of the coordinates. Therefore, the closer the percentage is to Pi, the more random the binary sequence is. The last test is serial correlation test which indicates how dependant each byte in the binary sequence on its previous byte. For a random binary sequence, its serial correlation should be close to zero.

The National Institute of Standards and Technology (NIST) test suite is a commonly used randomness test for evaluating random number generators. NIST tests were used in [7], [25], [26], for testing the security strength of their proposed cryptosystems. In the experiment, 10 binary sequences with a length of 10 kilobytes were generated from a symmetric neural cryptosystem with $M = 512$ and a batch size of 256,000. As a large batch size B results in a decrease in the Encryptor loss $\mathcal{L}_{E_b}$, making ciphertext more random. Due to the large batch size and limited memory of GPUs, only the Encryptor is trained in this experiment. The results from NIST tests are listed in the Table VII. The minimum pass rate for each statistical test is approximately 80% (8 out of 10), therefore, all tests listed in Table VII have passed. The P-values in Table VII are the uniformity tests for each individual test. For an individual test, the P-value>0.0001 indicates the probability of passing the test for all testing binary sequences is uniformly distributed across the interval [0,1) [27].

Frequency test represents how close are the appearances of 0s and 1s in the entirety of a binary sequence, whereas block frequency test only investigates the frequencies of 0s and 1s in small blocks of a binary sequence. Cumulative Sums test is a random-walk test, which converts 0s to -1s and sums equally divided parts of a binary sequence. If large sums occur too often for a binary sequence, it is not random. Runs test and longest run test investigate the total number of runs in a binary sequence, and a run is a succession of 0s or 1s. Different length of runs are tested to determine whether the oscillations between 0s and 1s occur too often or too rare. Rank test is to investigate the linear dependence among equally divided parts of a binary sequence. Fast Fourier Transform (FFT) test investigates the heights of the peaks in the Fourier transform of a binary sequences, if the peaks are too high and too close, the binary sequences have periodic patterns that can be exploited by attackers. Non-overlapping and overlapping Template Matching tests investigate the number of occurrences of a binary sequence in a sliding window that matches pre-defined non-overlapping or overlapping templates. Serial test investigates the occurrences of all possible overlapping fixed-length patterns across a binary sequence. Linear complexity test calculates linear feedback shift registers for a binary sequence, and longer registers indicate more randomness of the binary sequence. Passing all the NIST tests listed in Table VII proves that the security strength of the proposed neural cryptosystems is high, and the plaintext cannot be derived from ciphertext easily.

V. Conclusions

In this paper, we propose a novel light-weight IoT device authentication, encryption, and key distribution approach using neural cryptosystems and binary latent space. The proposed neural cryptosystems adopt three types of end-to-end encryption schemes, namely symmetric, public-key, and without keys. A series of experiments was conducted to test the performance and the security strength of the proposed neural cryptosystems, the results of which show that it is a promising security and privacy solution to next-generation large-scale IoT systems. Future work of the proposed approach might explore other ways to teach neural networks to hide plaintext in ciphertext. Furthermore, as the chi-squared test indicated that there was dependence among ciphertext, correlations among ciphertext should be further investigated.

References

[1] W. Kinzel and I. Kanter, “Neural cryptography,” in *Proceedings of the 9th International Conference on Neural Information Processing*, vol. 3, 2002, 1351–1354 vol.3.

[2] A. Klimov, A. Mityagin, and A. Shamir, “Analysis of neural cryptography,” in *Advances in Cryptology — ASIACRYPT 2002*, Y. Zheng, Ed., Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, pp. 288–298, ISBN: 978-3-540-36178-7.

[3] S. Cass, *Quickly embed ai into your projects with nvidia's jetson nano*, 2020. [Online]. Available: https://spectrum.ieee.org/geek-life/hands-on/quickly-embed-ai-into-your-projects-with-nvidias-jetson-nano.

[4] F. Manganiello, *Detecting people with a raspberry pi, a thermal camera and machine learning*, 2019. [Online]. Available: https://www.iotforall.com/raspberry-pi-thermal-camera/.

[5] TensorFlow, *Tensorflow lite for microcontrollers*, 2020. [Online]. Available: https://www.tensorflow.org/lite/microcontrollers.

[6] M. Abadi and D. G. Andersen, "Learning to protect communications with adversarial neural cryptography," *arXiv preprint arXiv:1610.06918*, 2016.

[7] A. Yayik and Y. Kutlu, "Neural network based cryptography," *Neural Network World*, vol. 2, no. 14, pp. 177–192, 2014.

[8] V. Sagar and K. Kumar, "A symmetric key cryptography using genetic algorithm and error back propagation neural network," in *2015 2nd International Conference on Computing for Sustainable Global Development (INDIACom)*, IEEE, 2015, pp. 1386–1391.

[9] V. Khavalko and A. Khudyy, "Application of neural network technologies for information protection in real time," in *2018 IEEE First International Conference on System Analysis and Intelligent Computing (SAIC)*, IEEE, 2018, pp. 1–4.

[10] I. Anikin, A. Makhmutova, and O. Gadelshin, "Symmetric encryption with key distribution based on neural networks," in *2016 2nd International Conference on Industrial Engineering, Applications and Manufacturing (ICIEAM)*, IEEE, 2016, pp. 1–4.

[11] R. Ramamurthy, C. Bauckhage, K. Buza, and S. Wrobel, "Using echo state networks for cryptography," in *International Conference on Artificial Neural Networks*, Springer, 2017, pp. 663–671.

[12] J. Blackledge, S. Bezobrazov, and P. Tobin, "Cryptography using artificial intelligence," in *2015 International Joint Conference on Neural Networks (IJCNN)*, IEEE, 2015, pp. 1–6.

[13] Y. Zhu, 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)*, IEEE, 2018, pp. 472–478.

[14] A. F. O. Gaffar, 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)*, IEEE, 2019, pp. 128–133.

[15] L. Zhou, J. Chen, Y. Zhang, C. Su, and M. A. James, "Security analysis and new models on the intelligent symmetric key encryption," *Computers & Security*, vol. 80, pp. 14–24, 2019.

[16] M. Coutinho, R. de Oliveira Albuquerque, F. Borges, L. J. Garcia Villalba, and T.-H. Kim, "Learning perfectly secure cryptography to protect communications with adversarial neural cryptography," *Sensors*, vol. 18, no. 5, p. 1306, 2018.

[17] S. Jhajharia, S. Mishra, and S. Bali, "Public key cryptography using neural networks and genetic algorithms," in *2013 Sixth International Conference on Contemporary Computing (IC3)*, IEEE, 2013, pp. 137–142.

[18] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, and Y. Bengio, *Binarized neural networks: Training deep neural networks with weights and activations constrained to +1 or -1*, 2016. arXiv: 1602.02830 [cs.LG].

[19] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[20] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, *et al.*, "Tensorflow: A system for large-scale machine learning," in *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI16)*, 2016, pp. 265–283.

[21] R. Chien, "Cyclic decoding procedures for bose-chaudhuri-hocquenghem codes," *IEEE Transactions on information theory*, vol. 10, no. 4, pp. 357–363, 1964.

[22] M. D. Macleod, "14 - coding," in *Telecommunications Engineer's Reference Book*, F. Mazda, Ed., Butterworth-Heinemann, 1993, pp. 14-1 - 14–13, ISBN: 978-0-7506-1162-6. DOI: https://doi.org/10.1016/B978-0-7506-1162-6.50020-4.

[23] S. Behnia, A. Akhavan, A. Akhshani, and A. Samsudin, "Image encryption based on the jacobian elliptic maps," *Journal of Systems and Software*, vol. 86, no. 9, pp. 2429–2438, 2013.

[24] Y. Sun and B. Lo, "Random number generation using inertial measurement unit signals for on-body iot devices," 2018.

[25] S. Toughi, M. H. Fathi, and Y. A. Sekhavat, "An image encryption scheme based on elliptic curve pseudo random and advanced encryption system," *Signal processing*, vol. 141, pp. 217–227, 2017.

[26] Ü. Çavuşoğlu, S. Kaçar, I. Pehlivan, and A. Zengin, "Secure image encryption algorithm design using a novel chaos based s-box," *Chaos, Solitons & Fractals*, vol. 95, pp. 92–101, 2017.

[27] M. Sys, Z. Riha, *et al.*, "Nist statistical test suite–result interpretation and optimization," 2015.