Hindawi
Wireless Communications and Mobile Computing
Volume 2022, Article ID 9584894, 13 pages
https://doi.org/10.1155/2022/9584894

Research Article

Convolution Neural Network-Based Sensitive Security Parameter Identification and Analysis

Hyunki Kim[1], **Donghyun Kim**[2] **and Okyeon Yi**[3]

[1]*Department of Financial Information Security at Kookmin University, Seoul 02707, Republic of Korea*
[2]*Department of Computer Science at Georgia State University (GSU), Atlanta, GA, USA*
[3]*Department of Information Security Cryptology and Mathematics at Kookmin University, Seoul 02707, Republic of Korea*

Correspondence should be addressed to Okyeon Yi; oyyi@kookmin.ac.kr

Received 9 December 2021; Revised 2 February 2022; Accepted 10 February 2022; Published 1 April 2022

Academic Editor: Liran Ma

When the logic of a pseudo-random number generator is followed by standards, the security of the generator depends on the entropy sources used in the seed construction, which are constructed through noise sources; thus, the noise sources are extremely important in cryptographic applications. The operation process of a cryptographic random number generator can be divided into an entropy collection stage and a pseudo-random number generation stage. To ensure the security, the noise sources used to construct the entropy source must be securely collected. If a cryptographic analyst physically acquires a cryptographic module in the form of a black box, it may be possible to predict the cryptographic random number to be used in the future by analyzing the random number generation process of the module. Thus, even if the cryptographic algorithms and protocols are securely designed and implemented without vulnerabilities, if the random number generation process is analyzed, the cryptographic system may be exposed. Noise sources can be identified through information generated when they are collected. If the noise sources can be identified during the stage of collecting sources that provide unpredictability to the cryptographic module during random number generation, future values may be predicted according to the data characteristics. We identify noise sources that are used as entropy sources with our convolution neural network model. Therefore, we establish attack scenarios in which random numbers can be analyzed by identifying data used in the model learned through this study when a random cryptographic module is obtained.

1. Introduction

Modern cryptography is applied under the assumption that secure random numbers are used. Random numbers are used as security parameters in cryptographic modules or cryptosystems such as encryption keys, nonces, and cryptographic salts [1–3]. Therefore, if the security of random numbers is not guaranteed, the entire cryptographic system is vulnerable. Because the vulnerability of the algorithm for generating random numbers significantly affects the security of the module or the entire system, it is more useful than the vulnerability of the encryption algorithm itself in terms of attacking the actual cryptographic system. That is, the random number is an essential element for the secure use of cryptographic functions such as confidentiality, authentication, access control, and nonrepudiation [4].

A "random number" used for cryptographic purposes has to be generated from the output of a cryptographic random number generator consisting of an entropy source collection stage and a pseudo-random number generation stage. The entropy source is the data used to generate the seed, which is the input to the deterministic random number generator.

Assuming that a secure deterministic random number generator (also known as the pseudo-random number algorithm) is used, the security of the cryptographic random number generator depends on the "unpredictability of the entropy sources" [5]. Therefore, to evaluate its security, it is necessary to verify the security of the input (seed) combined with the entropy sources. Typically, the security of a noise source to be used as an entropy source is verified based on entropy, independence, and statistical

tests. In addition, according to international random number-related standards, the seed, which is the input of the deterministic random number generator, has to be used in combination with various noise sources, and thus, the security of the noise sources is extremely important [4, 5].

By contrast, cryptographic random number generators can be classified into an entropy collection stage and a pseudo-random number generation stage, as mentioned above. To ensure the security of each stage, an evaluation of the noise sources used in the construction of the entropy sources applied to generate the pseudo-random number should be conducted [6]. If a cryptographic analyst physically acquires a cryptographic module in the form of a black box whose inside is unknown, it may be possible to predict the cryptographic random number to be used in the future by analyzing the random number generation process of the module. Then, even if all cryptographic algorithms and protocols are securely designed and implemented without vulnerabilities, all cryptographic systems using that random number generation process may be exposed to vulnerabilities. Accordingly, we present methods for identifying various entropy sources used as inputs to generate cryptographically secure random numbers by analyzing them with convolution neural network (CNN) models. This paper shows that CNN models can identify various data acquired during the collection stage of various entropy sources of cryptographic modules. Therefore, we can establish attack scenarios that can analyze a random number by identifying the entropy sources used in the model learned through this study as a cryptographic module under specific environments.

We establish three attack scenarios by considering the case where the cryptographic module to be analyzed is physically obtained and the case where it is not. If the attacker has physically acquired the module, the security parameters are identified by observing the output values. Otherwise, they are identified by observing the power consumption of the module.

The contributions to this paper are as follows.

1.1. Proposal of Cryptographic Module Analysis Methodology. This paper presents a method to analyze cryptographic modules from a new perspective. The main directions of the existing cryptographic module analysis methods were full investigation of implementation vulnerabilities, memory leak inspection, and reverse engineering. However, a new direction can be added to the analysis method through the CNN model-based security parameter identification methods proposed in this paper.

1.2. Suggestion of the Security Parameter Identification Model according to the Output Values of Cryptographic Modules. We accumulate several sensitive security parameters generated by software modules of Windows 10 and image them. Then, the security parameters are identified by predicting the images with the CNN model. As a result of that, we establish an attack scenario.

1.3. Suggestion of the Security Parameter Identification Model according to the Power Waveform Generated by the Cryptographic Module. We image the power consumption waveforms when security parameters are generated in the firmware cryptographic module in the Arduino board. And then, by predicting the images with the CNN model, parameters are identified. As a result of that, we establish an attack scenario.

2. Background

2.1. Random Number. Cryptographic random numbers are used as various security parameters. They must satisfy the unpredictability, unbiasedness, and interbit independence and provide more than the security strength recommended in [1, 7]. A deterministic random number generator (DRBG) is used to ensure unbiasedness and independence between bits, and various entropy sources (noise sources) are collected and mixed to satisfy the unpredictability and security strength recommended by the country and then input into the DRBG [6, 8].

2.2. Randomness. Randomness in cryptography (and statistics) means that it is difficult to find predictability or patterns of certain events that are randomly selected within a defined range. A random number is equivalent to the result of tossing a coin in an unbiased and fair manner. Each side of this coin can be considered 0 or 1 bit, and once tossed, the probability of obtaining a 0 or 1 is the same, with each being executed independently [2, 3]. Therefore, tossing a coin in succession can be considered the result of a perfect random number generator. Because all parts of a sequence are independent of each other, already generated parts cannot affect the generation of the next sequence, and it is also impossible to predict the next sequence from the current sequence. If a sequence is independent for each bit and has the same probability (also known as independent and identically distributed), it is said to be random.

2.3. Unpredictability. In an environment using cryptography, random numbers should never be predictable. If even a slight predictability will result in a different outcome. Depending on the situation, it is possible to attack the security protocol by predicting parameters such as encryption key, nonce, and salt. In addition, the ciphertext of a secure standard algorithm can be decrypted with the predicted key. For this reason, countries set strict standards to prevent random numbers from being predicted. Random numbers generated through cryptographic algorithms should not only be unable to predict future values based on current values, they should also be unable to infer past records based on such values. In general, random number generation algorithms are open to the public, and thus, the input values of the algorithms have to remain secret to ensure their unpredictability [2, 3].

2.4. Security Strength (Bits of Security). Security strength is closely related to the number of operations required to break a cryptographic algorithm or system, expressed in bits, and the National Institute of Standards and Technology (NIST)

standard stipulates that one should be selected from the set of {80, 112, 128, 192, 256} the selected [7]. A security strength of 80 bits should not be used, however, as it is no longer considered sufficiently secure. In addition, it is currently stipulated that more than 112 bits should be provided.

2.5. Random Number Generation. Because cryptosystems frequently use random numbers, they should be able to generate them extremely quickly. It is impossible to obtain a random number for cryptographic use without a specific algorithm, and even if the output value changes every time without an algorithm and it is possible to generate an unpredictable value (true random number), the speed is sufficiently slow to avoid satisfying the availability. For this reason, in the cryptography field, a deterministic algorithm is used to generate random numbers at a high speed [4].

The algorithm used in this situation is called DRBG, which receives seeds and generates random numbers according to the determined logic. Therefore, when the same seed is set in the same DRBG, these two algorithms output random numbers that match each other exactly. Because of this characteristic, if the user of an algorithm applies predictable values as seeds, an attacker can accurately determine the random number that the user generates. Thus, the seed that initializes or updates the DRBG must be different each time to be unpredictable. In general, these consist of noise sources that are commonly obtained in the operating environment [6]. Because most noise sources have entropy bias, it is inappropriate to use the collected noise source as a seed without any significant change. NIST and the Telecommunications Technology Association (TTA) recommend applying conditioning to equally distribute the biased entropy. The lower bound of the security of the DRBG depends on the entropy provided by the noise sources and the way in which the seed is constructed. It is therefore necessary to configure the seeds with entropy sources that guarantee the target security strength of the cryptographic module [9, 10].

2.6. Noise Sources, Entropy Sources, and Seed. When using the DRBG as a secure standard algorithm, its security depends entirely on the seeds used as input. Therefore, the composition of the seed is extremely important. What is needed are noise sources or entropy sources, because these must consist of values that change unpredictably and nondeterministically. The "noise source" encompasses a function or device that generates nondeterministic data, and the "entropy source" includes a function or device that combines the noise source and an algorithm (as the conditioning) that mitigates its poor characteristics [1, 8].

As shown in Figure 1, noise sources have various collection methods (digital or analog data) and types (e.g., function calls and sensing). In general, as noise sources that can be collected by the operating system, aspects such as the running time, mouse cursor coordinates, keyboard stroke, disk status, and interrupt request information can be used. The entropy of nondeterministic functions provided by the operating system may change depending on the environment, such as the current state of the operating system, user intervention (e.g., mouse/keyboard), and the number of active background processes. Therefore, it is recommended to use not only one type of noise source provided by the operating system but also various types together [4]. If a certain noise source has entropy characteristics, it can be used as a component of the entropy source for seed construction. If not, however, it cannot be used to configure the entropy source but can be used for the purpose of supplementing the stability of the seed.

In the server and PC level equipment, it is not difficult to accumulate more entropy than the security level because there are many noise sources that can be used as entropy sources. However, small devices such as Internet of Things (IoT) devices with firmware only or sensors that cannot install an operating system lack the ability to accept user intervention or generate unpredictable elements. Therefore, the available noise sources are extremely limited [11–13].

3. Related Work

3.1. Data Identification Using Deep Learning. Deep learning refers to machine learning technologies that construct a model to have a large number of neural layers for learning a pattern recognition problem or feature points. This is an artificial neural network (ANN) technology consisting of several hidden layers between an input layer and an output layer. With artificial intelligence (AI) and ANNs in the spotlight, deep learning is used in various fields. Representative neural network structures include an autoencoder [14] and a restricted Boltzmann machine (RBM) [15], convolutional neural network (CNN) [16], and recurrent neural network (RNN) [17]. An autoencoder and RBM are models presented in the early days when deep learning technology was proposed and have the advantage of being easy to use for self-learning. However, they exhibit limited performance owing to their characteristics. Most currently used deep learning systems are based on a CNN or an RNN, and although they require supervised learning, they show a more powerful performance and a wider range of applications. CNNs are becoming an essential technology in the field of image signal processing or computer vision [18, 19], and RNNs have shown a good performance in speech signal processing and speech recognition. CNNs are neural networks designed using the concept of the receptive field of human visual neurons [16]. They have convolutional layers, pooling layers, and rectified linear units (ReLU) as key elements. They are characterized by kernels that take inputs and outputs in a specific form and represent the weights as small-sized filters. That is, regardless of the number of dimensions of the input data, if the kernel size is small, the neural layers can be defined with an extremely small number of weights. The pooling neural layer reduces the size of the data by summarizing several output values, and through this process, it can eliminate the distortion of the input data. A ReLU is a nonlinear neuron with ramp function characteristics and has the effect of solving the computational burden of the "sigmoid" function and the disappearance of gradients in the backpropagation algorithm.

Although the concept of a CNN was proposed at the end of the 20th century, it was grafted with the deep learning

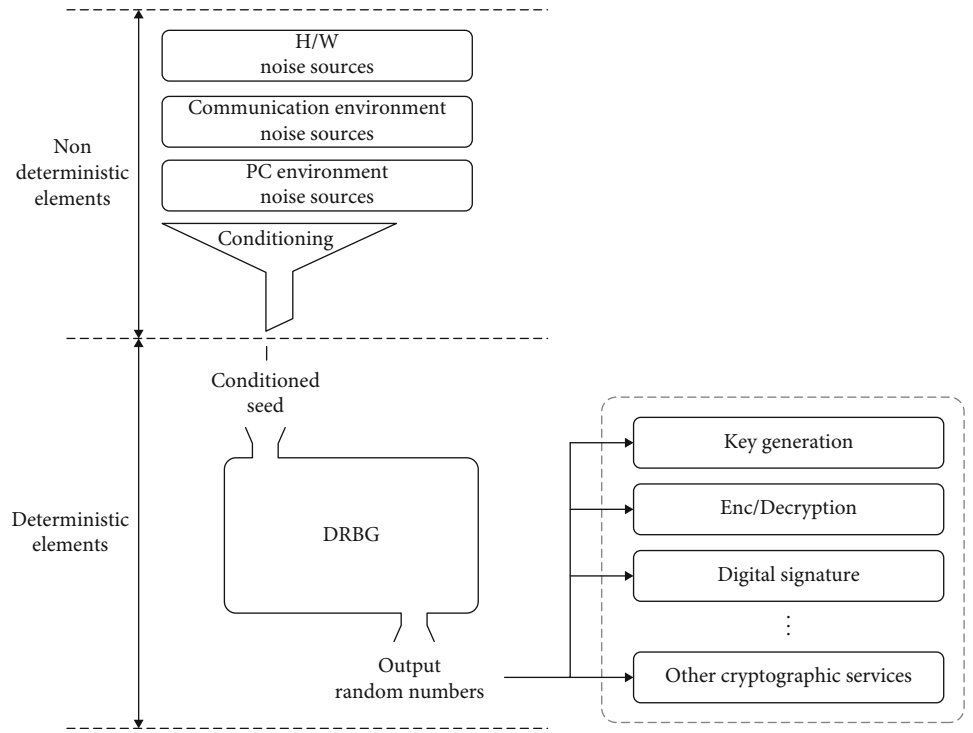

Figure 1: Roles of randomness. In the environment in which the cryptographic module operates, nondeterministic data are variously combined and passed through deterministic algorithms. The generated random numbers are then used as different security parameters.

Table 1: Windows OS noise source providing functions and their information.

Collected noise source information	Function name
Windows random number generator	CryptGenRandom()
System configuration information	GetSystemInfo()
Operating system version information	GetVersionEx()
Current process ID	GetCurrentProcessId()
Current thread ID	GetCurrentThreadId()
Memory state information	GlobalMemoryStatus()
Mouse cursor position	GetCursorPos()
CPU clock accumulated after booting	QueryPerformanceCounter()
CPU clock generated per second	QueryPerformanceFrequency()
Process heap handle	GetProcessHeap()
Elapsed time since boot	GetTickCount()
Create GUID	CoCreateGuid()

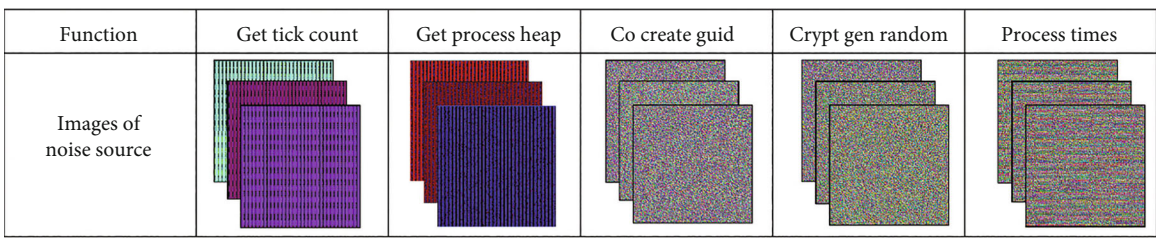

Figure 2: Samples of noise sources. The noise source functions were called several times to form each bit stream, and through them, ".bmp" files of 255×255 in size were made. Although GetTickCount and GetProcessHeap are easy to identify because of their distinct characteristics, it is difficult to tell the difference between CoCreateGuid, CryptGenRandom, and ProcessTimes.

Table 2: Estimated entropy values for noise sources by byte index through SP800-90B. Target noise sources are mainly used as entropy sources for software cryptographic modules in Windows 10. The smaller the entropy per byte (sum total/number of bytes), the more patterns appear in the image, and vice versa; it becomes increasingly difficult to recognize the pattern. The noise sources CryptGenRandom and CoCreateGuid, commonly used in Windows's software cryptographic modules, have quite high entropy values and do not differ much. Therefore, even in their images, it seems indistinguishable by the eyes.

Target noise		GetTickCount	GetProcessHeap	CoCreateGuid	CryptGenRandom	GetProcessTimes
Byte index	0	0.303	0.000	7.863	7.883	7.882
	1	0.027	0.000	7.885	7.881	7.008
	2	0.002	7.876	7.884	7.877	0.587
	3	0.000	7.852	7.882	7.882	0.001
	4		7.882	7.878	7.888	
	5		0.996	7.880	7.871	
	6		0.000	7.886	7.865	
	7		0.000	3.975	7.880	
	8			5.939	7.879	
	9			7.881	7.861	
	10			7.869	7.879	
	11			7.883	7.874	
	12			7.862	7.864	
	13			7.880	7.889	
	14			7.883	7.879	
	15			7.881	7.865	
Sum total of H		0.332	24.606	120.211	126.017	15.478
Average of min-entropy H		0.083	3.076	7.513	7.876	3.870

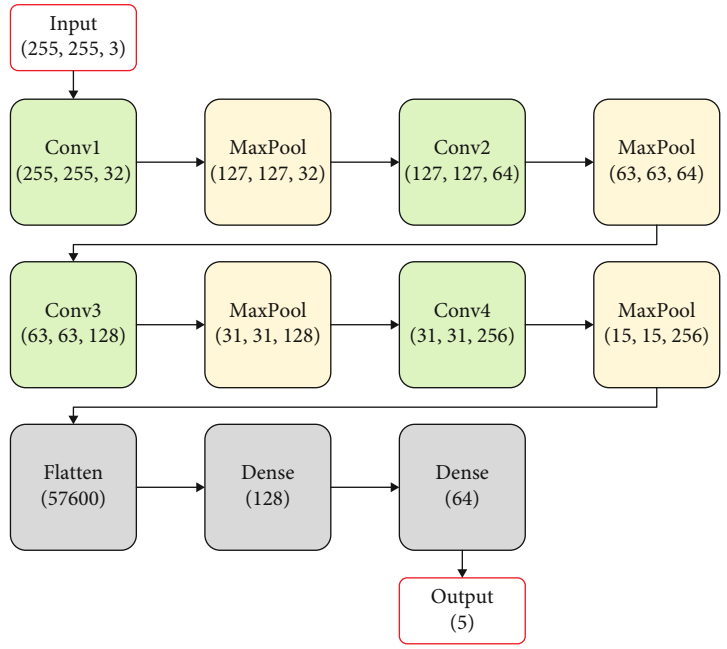

Figure 3: Structure of $CNN_{NSBased}$. It consists of four Conv2D layers, and Maxpooling is applied to each layer. The shape of the data output from each step of the model is indicated in parentheses.

paradigm during the 2012 ImageNet challenge contest and has become the mainstream of both deep learning and machine learning. Until 2011, a large number of images were classified using the existing machine learning method, and with the 2012 release of the CNN-based AlexNet [20] , it has achieved significant results, becoming the mainstream tool in image-related problems. CNN models proposed after 2012 have had an increasing number of neural layers, and

Table 3: Hyperparameters of CNN models.

Hyperparameter	$CNN_{NSBased}$	$CNN_{TraceBased}$
Filter size	{32, 64, 128, 256}	{4, 8, 16, 32}
Epoch	20	15
Output size (number of classes)	5	4
Batch size	100	100
Kernel size	5×5	5×5
Kernel initializer	he_normal	he_normal
Conv strides	(1, 1)	(1, 1)
Pooling strides	2	2
Output activation type	Softmax	Softmax
Optimizer	rmsprop	rmsprop
Loss function	categorical_crossentropy	categorical_crossentropy
Padding	Same	Same
Pooing sizes after each conv layer	(2, 2)	(2, 2)

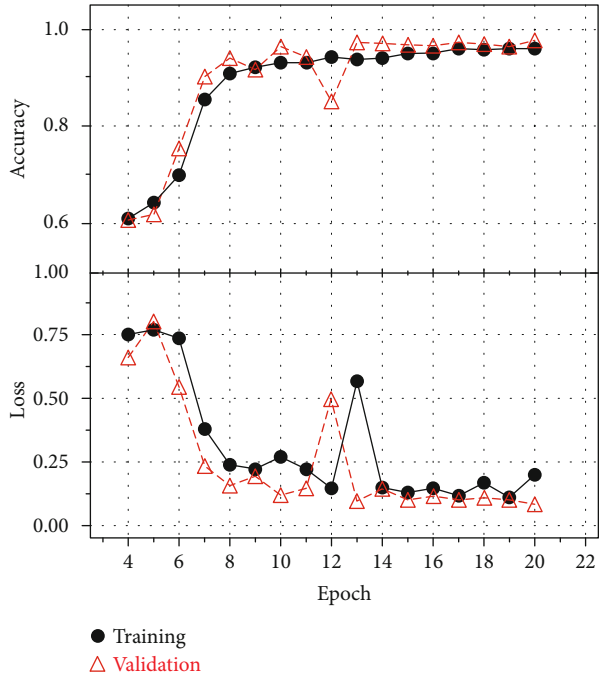

Figure 4: Result of identifying noise sources of Windows 10. In up to 8 out of 20 epochs in total, the accuracy sharply increases to approximately 0.90, and the loss rapidly decreases to approximately 0.2. After that, it gradually increases during the period and finally reaches 0.97. From epoch 21, the training loss decreases and the validation_loss tends to increase, resulting in an overfitting, and thus, the training is stopped at epoch 20.

through this process, techniques for applying many neural layers into a single CNN have been applied. In 2013, the network-in-network (NIN) concept [21] was proposed in which the filter kernel is composed of a neural network rather than a set of weights. In addition, in 2014, VGG [22] and GoogLeNet [23], which apply the technique of stacking many small convolutional layers rather than stacking few large convolutional layers, were proposed. In 2015, ResNet was proposed [24], which reduces the number of computations by learning the difference between outputs and inputs, and constructs a deeper neural network. The approach recorded an image classification error rate of 3.5% in the ImageNet challenge competition and was evaluated as achieving a higher performance than the human cognitive ability of 5%. Meanwhile, a neural network in the form of a combination of GoogLeNet and ResNet has been released, and the structure of CNNs will continue to become deeper and more complex in the future [25].

3.2. Security Evaluation of Cryptographically Secure Random Number. Methods used to evaluate the security of cryptographic random numbers can be classified into statistical randomness tests and stochastic entropy estimations [26]. In general, a statistical randomness test is a method for checking whether the final output of a random number generator is distinguished from ideal random numbers, and the entropy estimation method estimates the minimum entropy of a noise source providing unpredictability [27]. The security evaluation of cryptographic random numbers is being studied mainly by major standard organizations such as NIST in the US and the Federal Office for Information Security (BSI) in Germany [28, 29]. Appropriate probabilistic entropy estimation methods can theoretically prove the security of the entropy source. However, they are based on the assumption that the target of the estimated source and their behavior are both known [6, 27]. However, the statistical randomness test is unsuitable for discriminating noise sources, and it only judges whether the data are random numbers by focusing on the statistical properties regardless of the entropy of the target source.

4. Materials and Methods

4.1. Image Data Identification of Noise Sources Based on CNN. The final goal of this study is to identify random noise sources generated by the black box cryptographic module

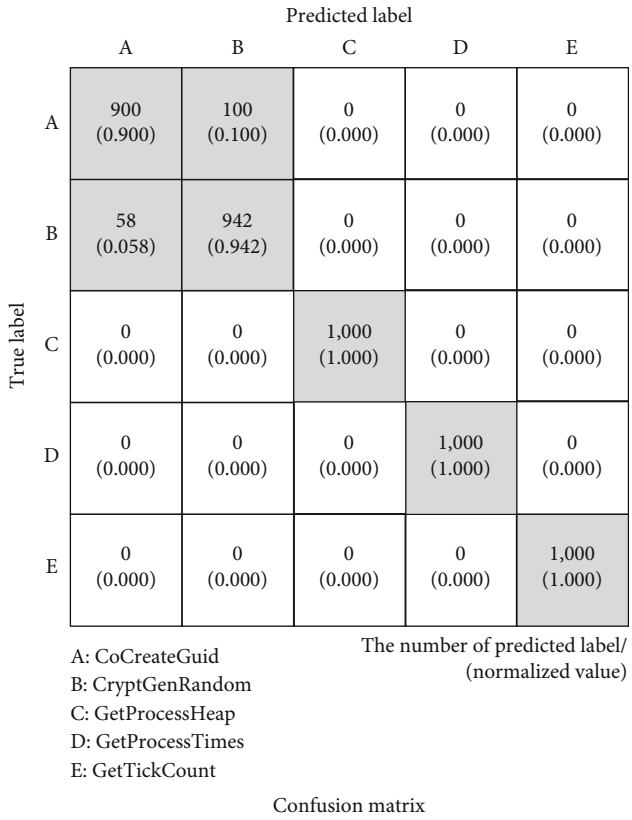

FIGURE 5: Confusion matrix for $CNN_{NSBased}$. This model correctly identified GetProcessHeap, GetProcessTimes, and GetTickCount for each 1,000 pieces of data. Although incorrect answers existed for CoCreateGuid and CryptGenRandom, they were identified with high probability. Each value in the table means the number of predicted labels and their normalized value.

whose interior is unknown when using a trained model. To identify the noise source functions used in the acquired module, the model has to be trained with the noise source values provided under various environments in advance. Because this paper focuses on identifying the noise sources generated by the cryptographic module, we start by examining the noise sources that can be obtained in a general PC environment and are mainly adopted in a cryptographic module.

4.1.1. Selection and Collection of Target Data. First, the target of the training is noise sources commonly used as entropy sources in the Windows 10 OS environment. Table 1 shows a part of the Windows noise source functions provided [1].

Each bit stream was made by repeatedly calling some of these functions, which were imaged and used as training data for the model. This can be visualized by converting the noise source bit streams collected several times into a bitmap form. As shown in Figure 2, there are noise sources whose patterns can be distinguished even with the naked eye (GetTickCount() and GetProcessHeap()), whereas others are difficult to find patterns (e.g., CoCreateGuid() and CryptGenRandom()). When operating the cryptographic module in various environments, even if digital noise sources generated by software or analog signals generated from hardware are gathered, any noise source data must also be digitized because they must eventually go through the logic operation of the algorithm, and thus, the bit sampled per noise source is determined within a finite range. Then, the maximum and minimum values at which the noise source is output are determined, and accordingly, when the same type of noise source is repeatedly collected several times, a specific pattern may be repeated. This feature becomes more prominent when the bit stream collected several times is imaged. Therefore, to detect patterns that humans are unable to distinguish, a CNN is applied to an image analysis of various noise sources to identify them. To train the model, a total of five noise sources (CoCreateGuid(), CryptGenRandom(), GetProcessHeap(), GetTickCount(), and GetProcessTimes()) were selected for analysis. For this purpose, 50,000 samples of $255 \times 255 \times 3$ size (width × length × number of channels) were generated for each noise source. A total of 25,000 samples (50% of the total) were used as training data, and 20,000 samples (40%) were used as validation data. Finally, 5,000 samples (10%) were used for a prediction test to check the accuracy of the trained model.

(1) CryptGenRandom. This is a random number generator provided by Microsoft Windows that generates cryptographically random data; however, a vulnerability in the function was discovered in 2007 [30, 31], and it can no longer be used as a random number generator. However, because a different value is generated every time the function is called, the data are generally used as a noise source [4].

(2) CoCreateGuid. This creates a globally unique GUID each time it is called.

(3) GetProcessHeap. When this function is called, the number of heap handles that are active in the calling process is returned.

(4) GetTickCount. When this function is called, the elapsed time in milliseconds since the system was started is returned.

(5) GetProcessTime. Various types of time can be returned, such as the start or end time of the process being used and the time the process was executed in the kernel mode.

Meanwhile, Table 2 shows the min-entropy values of the target noise sources. These are the results of the entropy estimation software [6] published in 2020 by NIST. This tool has limitations that the size of one sample can only estimate the entropy from the minimum 1-bit to the maximum 1-byte (8-bit). Since all of the target noise sources are larger than 1 byte, the final entropy "H" was determined by obtaining the entropy corresponding to the byte position of each data and adding all the values. For example, if there are a total of 3 bytes with Noise = noise[0]‖noise[1]‖noise[2], each $H_{\text{noise}[0]}$, $H_{\text{noise}[1]}$, and $H_{\text{noise}[2]}$ is derived through the tool. Then, the final entropy is obtained as $H_{\text{Noise}} = H_{\text{noise}[0]} + H_{\text{noise}[1]} + H_{\text{noise}[2]}$.

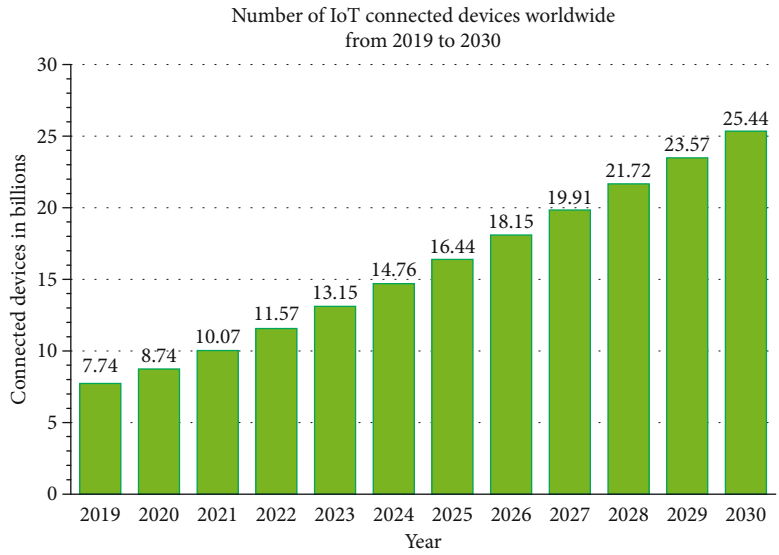

Figure 6: Number of IoT-connected devices worldwide from 2019 to 2030. It is predicted that the global use of IoT devices will nearly triple within 10 years.

Table 4: Functions that are supported by the Arduino board.

Category	Deterministic functions	Nondeterministic functions
Digital I/O	digitalRead()	
	digitalWrite()	—
	pinMode()	
Analog I/O	analogReference()	analogRead()
	analogWrite()	
T5ime	delay()	micros()
		millis()
Random numbers	random()	—
	randomSeed()	
Interrupts	—	timer0_overflow_count

4.1.2. Model Training. As shown in Figure 3, this noise source-based CNN ($CNN_{NSBased}$) goes through Conv2D four times, Maxpooling four times, and Dense three times. The filter consists of 32, 64, 128, and 256 square pixels for each conv layer, and Maxpooling is applied to reduce the number of features that can occur every time each layer is passed by half. In addition, ReLU is applied to the activation function of Conv, and in the last density layer, a softmax function is adopted to suit single index multiclassification. Moreover, RMSprop and categorical_crossentropy are used in the optimizer and loss functions, respectively, to construct a model. The model was trained to identify the noise source image through a total of 20 epochs with a batch size of 100. The details of the hyperparameters of this model are shown in Table 3.

4.1.3. Analysis of Results. As a result of training the model with images of entropy sources, the patterns of bit strings that are difficult to identify are distinguished with the naked eye with high accuracy. To check the possibility of identifying the entropy sources of this CNN model, as shown in Figure 2, bit strings (GetTickCount(), GetProcessHeap()) that are easy to detect with the naked eye and bit strings (CryptGenRandom(), CoCreatedGuid(), and ProcessTimes ()) that are difficult to identify were used as target data for learning and verification.

Assuming that the accuracy of choosing the correct answer by randomly selecting one out of five classes is 20%, the identification result of the trained model is shown in Figure 4. The identification accuracy started to increase from epoch 8 and reached approximately 97.24% at epoch 20. An overfitting occurred when epoch 21 was passed, and learning was stopped at epoch 20.

To evaluate this trained multiclass identification model, a dataset of 5,000 uncontaminated through model training was used for the prediction test. Figure 5 shows the normalized confusion matrix for the multiclass identification model results.

The results of the prediction test indicate that this model can identify noise sources with consistently high precision for GetProcessHeap, GetProcessTimes, and GetTickCount. This indicates that there are feature points that can be distinguished by the model between each noise source class. In addition, there is a percentage of instances with a misclassification between CoCreateGuid and CryptGenRandom. From this result, although the model can distinguish the two, it is inferred that their distributions are extremely similar.

Meanwhile, the $F1$ score can be derived by calculating the precision and recall, which are performance evaluation indicators of the identification model, based on the values in Figure 5. The precision and recall of CoCreateGuid (CCG) are $\text{precision}_{CCG} = 900/(900 + 58) = 0.94$ and $\text{recall}_{CCG} = 900/(900 + 100) = 0.90$, respectively, whereas those of CryptGenRandom (CGR) are $\text{precision}_{CGR} = 942/(100 + 942) = 0.90$ and $\text{recall}_{CGR} = 942/(58 + 942) = .94$, and

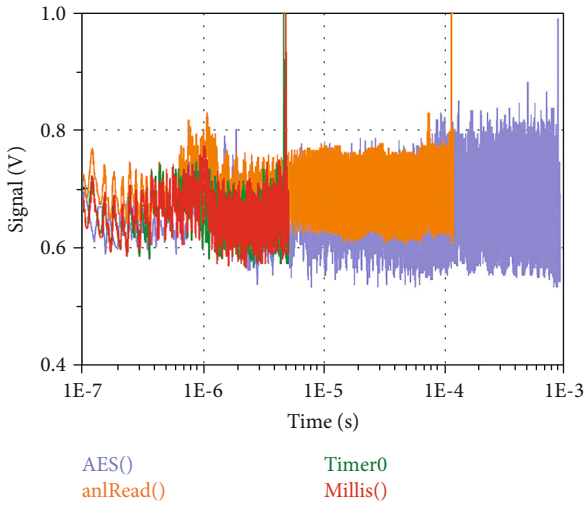

Figure 7: How to encode input for use. This is a graph in which the time of the waveform is expressed as a logarithmic scale. AES, analogRead, millis, and timer0_overflow_count (called Timer0) have the longest running times in order. Because millis and timer0 use similar clock-related interrupts, they tend to have similar shapes.

Table 5: Number of valid points in trace.

Target noise function	Total number of points in trace	Number of valid points between trigger signals
AES()	100,000	About 86,690
analogRead()	60,000	About 54,900
millis()	6,000	About 4,700
timer0_ overflow_ count	6,000	About 4,570

those of the other three classes are 1000/1000 = 1.00. The $F1$ score derived from these indicators is proximately 0.97, and thus, the model effectively identifies the noise sources.

4.2. Identification of Power Waveform Data When Noise Sources Are Generated Based on CNN. As shown in Figure 6, IoT devices are used worldwide from industrial control to consumer markets, and their number is expected to nearly triple from 8.74 billion units in 2020 to over 25.4 billion units in 2030 [32]. In particular, among the IoT devices available in 2021, Raspberry Pi and Arduino board occupy the first and second places with 44% and 28% shares, respectively, in the industrial control sector [33]. Industrial IoTs often share data between multiple parties [34, 35]. In particular, among the devices, they may upload or share privacy and sensitive information. At this time, security protocols are carried out to prevent inference attacks, etc. In this process, IoT requires random number generation [36, 37].

Even if some cryptographic analysts have not yet physically acquired a cryptographic module whose internals are unknown, if the power consumption waveform generated when the module is operating can be determined from a distance, the module can be analyzed through such waveforms [38, 39]. When applying specific functions in any module or device, the generated power consumption waveforms all differ. Some of these specific functions also include the process of generating important security parameters (resulting from noise-source generating functions). The model identified in this section is the power trace generated when generating sensitive security parameters (entropy sources) on an Arduino board. If an attacker can catch the power waveforms of a target board (e.g., a cryptographic module that has not been physically seized) from a distance, the attacker can reveal the types of security parameters used by the device and further identify them.

4.2.1. Selection and Collection of Target Data. The target board of our experiment is an Arduino Uno; in addition, the MCU is an ATmega328P based on an 8-bit operator, the clock speed is 16 MHz, and the board operates at a voltage of 5 V. Most IoT devices provide only limited functions that can only perform specific tasks to minimize the hardware volume, power, and memory, among other factors. Therefore, the random elements are so limited that they cannot be compared with the PC environment, and thus, there are few available functions. The noise sources including random elements that can be used on the Arduino board are at most functions that read the input voltage from the analog pin and the built-in timer-related functions. Thus, as a training target for the CNN model used in this experiment, power waveforms generated by the target board (when calling the noise source functions and encryption algorithms) were applied.

Table 4 shows some of the basic functions provided by the Arduino board [40]. Since the functions supported are extremely limited, good noise sources (high entropy) cannot be obtained unless there are added-on modules and sensors. Various sensors such as gyroscope, position, inertia, vibration, and acceleration can be added to the Arduino board according to the application used in the industry. There is also a method of digitizing analog values generated from newly added devices and using them as noise sources. However, since the combination of sensors added-on varies extremely depending on the board usage in each industry, this paper deals with only the functions that are basically supported. The method of obtaining a noise source by adding a module other than Arduino is out of scope of the experiment. The power consumption that occurs when three out of the four functions whose resulting value is not deterministic (analogRead(), millis(), and timer_overflow_count) are operated was collected. The representative encryption algorithm AES() was also collected for comparison [41]. In addition, the power consumption generated at that time was collected. First, in the target board, after two channels (waveform and trigger channel) are connected to the oscilloscope, a rising edge is activated just before the target algorithm operates, and a falling edge is activated immediately after operation. The learning target data of the model is collected by extracting the waveform during the trigger signal. Figure 7 is the graph, which is expressed on a logarithmic scale of the power consumption used when the target functions operate. And Table 5 shows the number of points

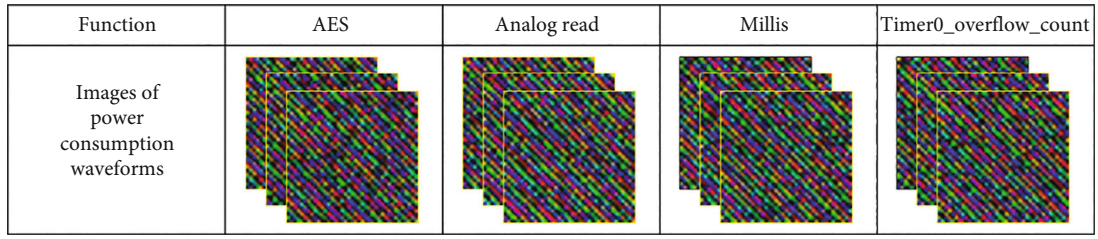

Figure 8: Image samples of each trace. Unlike when the output bit streams in Section 4.1 were imaged, the images of the traces were uncharacteristic. Therefore, it is impossible to identify with the naked eye.

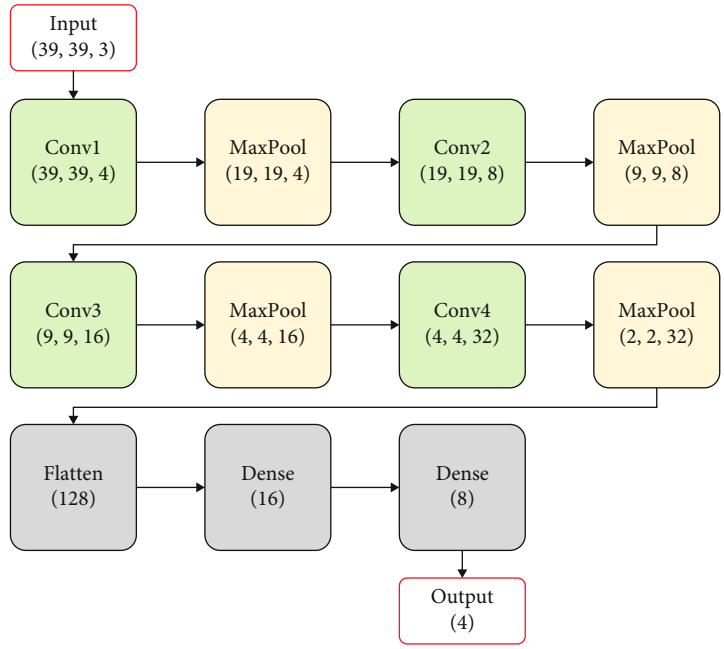

Figure 9: Structure of $CNN_{TraceBased}$. It consists of four Conv2D layers, and Maxpooling is applied to each layer. The shape of the data output from each step of the model is indicated in parentheses. The details of the hyperparameters of this model are shown in Table 3.

measured in the target power consumption waveform during the trigger signal. Because AES() and analogRead() perform many operations inside the corresponding algorithm, the running time is relatively long and there are numerous points in the trace. In addition, timer0_overflow_count and millis() require less running time and contain a small number of points compared to AES() and analogRead(). As shown in Figure 8, to train the power waveform identification model with the data of the same length, the number of the sampled data of each target noise function is determined by timer0_overflow_count because it has the minimum number of valid points during the pulse width of the trigger signal.

In addition, to train the waveform identification model in a similar way as the model $CNN_{NSBased}$, the extracted data were postprocessed as a square color bitmap file. Using $39 \times 39 \times 3 = 4{,}563$ (<4,570, which is the number of points in timer0) points for each waveform, a total of 10,116 data samples of size (39, 39, 3) (horizontal, vertical, and number of channels) were generated. Square bitmap files of color are represented as shown in Figure 8. We used 6,068 pieces (60% of the total) as training data and 3,032 pieces (30%) as validation data. In addition, 1,016 pieces (10%) were used for the prediction test to check the accuracy of the trained model.

4.2.2. Model Training. As shown in Figure 9, this CNN model ($CNN_{TraceBased}$) applies Conv2D four times, Maxpooling four times, and Dense three times, which are the same numbers as the noise source identification model $CNN_{NSBased}$. Because the pixel resolution of the image is 39×39, the filter is composed of 4, 8, 16, and 32 square pixels for each Conv layer to avoid exceeding the width and height of the image. For the same reason as the previous model, Maxpooling is applied to halve the features that can occur each time each layer is passed. The activation function of Conv applied ReLU, the last Dense layer adopted a softmax function suitable for single-index multiclassification, and the optimizer and loss functions constructed a model using RMSprop and categorical_crossentropy, respectively. It was trained to identify the image of the power consumption waveform through a total of 15 epochs with a batch size of 100.

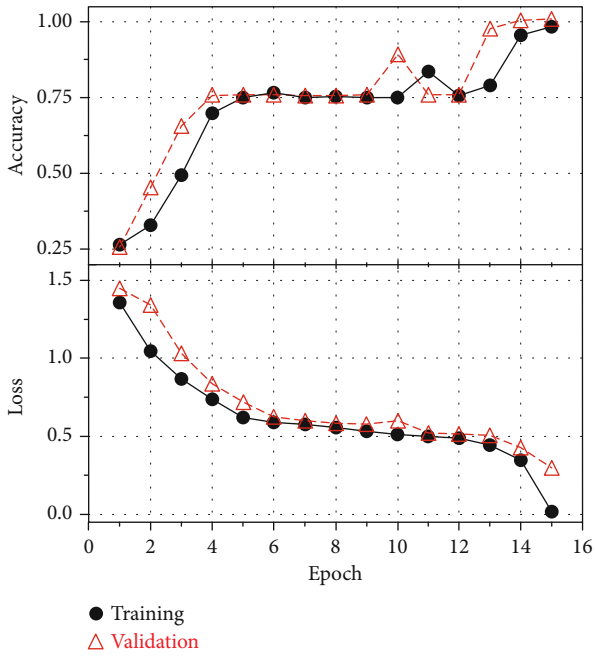

Figure 10: Results of identifying power consumption waveform of Arduino. For up to 4 out of the total 15 epochs, the accuracy sharply increases to approximately 0.75, and the loss rapidly decreases to approximately 0.75. After that, it gradually increases during the period and finally reaches 0.98. From epoch 16, the training loss decreases and the validation_loss tends to increase, resulting in an overfitting, and thus, the training was stopped at epoch 20.

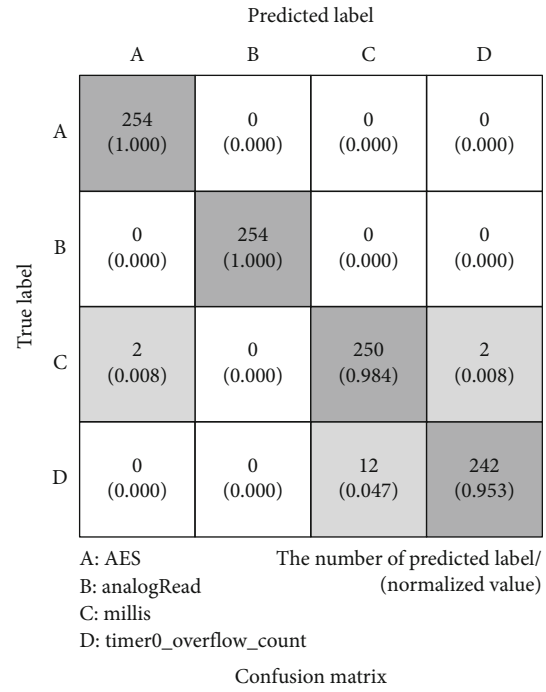

Figure 11: Confusion matrix for $CNN_{TraceBased}$. This model correctly identified analogRead for all 254 pieces of data. Although incorrect answers existed for AES, millis, and timer0, they were identified with high probability. Each value in the table indicates the number of predicted labels and their normalized value.

4.2.3. Analysis of Results. As a result of training the CNN model with the image of the power waveform generated when the noise source functions on the target board are called, the model distinguishes the pattern of bit strings that are difficult to identify with the naked eye with high accuracy. Here, AES() and analogRead() have distinct waveforms because they contain completely different logical operations internally, whereas millis() and timer0_overflow_count use similar types of clock interrupt handlers, and thus, the waveforms have similar shapes.

Therefore, the CNN model used in this experiment showed a high identification rate for AES() and analogRead() and accurately identified all prediction datasets corresponding to them. If one of the four classes is randomly selected and the accuracy of correcting the correct answer is 25%, the identification result of the trained model is shown in Figure 10. The identification accuracy gradually increased from epoch 3, and the accuracy was over 90% from epoch 14; in addition, learning was stopped at epoch 15 because an overfitting occurred from epoch 16 or higher. To evaluate this multiclass identification model, 1,016 datasets uncontaminated by training were used for the prediction tests. Figure 11 shows a confusion matrix of the results of this identification model.

The results of the prediction test indicate that this model can identify the target waveform with consistently high precision. Through this result, it can be seen that there are feature points that the CNN can distinguish among each power waveform class. In addition, there is a proportion of instances that misclassified between millis() and timer0_overflow_count. This leads to the fact that, although the model can identify the two, the power consumption according to the algorithm operation is extremely similar.

Meanwhile, based on the values in Figure 11, the precision, recall, and $F1$ score, which are performance evaluation indicators of the identification model, can be derived. The precision and recall of millis() are $\text{precision}_{\text{millis}} = 250/(250 + 12) = 0.95$ and $\text{recall}_{\text{millis}} = 250/(2 + 250 + 2) = 0.98$, respectively, and those of timer0_overflow_count (called timer0) are $\text{precision}_{\text{timer0}} = 242/(2 + 242) = 0.99$ and $\text{recall}_{\text{timer0}} = 242/(12 + 242) = 0.95$. The $F1$ score derived by calculating the other two classes in this way is approximately 0.98, and the model effectively identifies the noise sources.

4.3. Attack Scenarios. If some cryptographic analysts physically acquire a black box-type cryptographic module whose inside is unknown, they may try to analyze it in various ways, such as through reverse engineering, a brute force attack, and a side-channel attack, to understand the cryptographic mechanism through which the module operates. Under this situation, any information related to security, such as cryptographic algorithms, protocols, and cryptographic keys used within the module, can be targeted for analysis. This also includes sensitive security parameters (such as cryptographically random numbers or the noise sources needed to generate them). By contrast, even if the

analyst did not physically acquire the cryptographic module, if the power waveform generated when the module is operating can be caught from a distance, the module can be side-channel analyzed.

If analysts do not know what noise sources are used as entropy sources in the module, the noise source identification model of this study can be used to derive the types of noise sources used in the cryptographic module. If the noise sources used among the parameters derived from the target encryption module are revealed by physically acquiring or catching a remote signal, the noise sources can be reproduced or predicted. If the noise sources found are the time- and clock-related parameters, they can be monotonically increasing and repeat the same values within a finite range ($\{0,1\}^n$ at most).

Meanwhile, analysts can know the operating environments if the cryptographic module is physically obtained or if the target module of a remote analysis is a universally used public device. As a result, the rate of increase in clock noise sources in the environment can also be obtained without difficulty. Therefore, analysts can predict the output of a noise source that monotonically increases at a constant rate within a finite range, which can affect the security of the cryptographic system. Also, let us suppose that some IoT devices sense surrounding environmental information (e.g., temperature and illuminance) and use it as noise sources. Then, from a distance, the attacker already knows the noise source information of the IoT device (through the method of this paper), and as a result, he can manipulate the surrounding environment (such as the temperature of a heater and the brightness of a fluorescent lamp in a building) in which the target device is installed to a desired value.

4.4. Study Limitations. In order to obtain the identification effects presented in the attack scenarios section, the models must be trained in advance regardless of which cryptographic module is attacked. For the $\mathrm{CNN}_{\mathrm{NSBased}}$ model to be trained, the type of the attack target module must be clearly known. If the nationally verified cryptographic modules (e.g., CMVP, KCMVP, and JCMVP) are the target of the analysis, even if the module is not physically acquired, the device type of the module can be found on the website of the relevant verification program. However, when the target of analysis is a module used for private business or defense, it is not easy to know the type of device in advance. Meanwhile, to train the $\mathrm{CNN}_{\mathrm{TraceBased}}$ model, it must be able to remotely detect the power waveform when the attack target module is operating. This is an inherent limitation of all subchannel analysis methods. In addition, it is quite difficult to specify the waveform (e.g., the power/electromagnetic wave) of the module, and the sensors are expensive.

5. Conclusions

This paper presents a new perspective for analyzing cryptographic modules. We propose that the cryptographic system can be affected by identifying sensitive security parameters that are the starting point of cryptographic module security. We do not present a new deep learning identification model but show that anyone can analyze cryptographic modules through image identification models that are already commonly used in various fields. This is because even being able to identify a critical security parameter can lead to a fatal vulnerability.

In this paper, entropy sources, which are the basis of cryptographic random numbers, were analyzed by type through CNN image identification models. As a result, the existence of patterns that are difficult to detect with the human eye and the possibility of detecting and identifying entropy sources of unknown origin were presented.

As future studies, the following can be applied to improve the identification performance of deep learning for any noise source. The identification of noise sources in a Raspberry Pi, which has the largest market share among industrial IoT devices, suggests that additional problems are caused by such sources, and the internal configuration of the learning model should be changed. In particular, in order to improve the identifiability of security parameters, it can also be considered to apply the Gated Recurrent Unit (GRU) and Long Short-Term Memory (LSTM), which are models that have different identification methods from CNNs.

Data Availability

The data used to support the findings of this study are included within the article.

Conflicts of Interest

The authors declare that there is no conflict of interests regarding the publication of this paper.

Acknowledgments

This research was supported by the Republic of Korea's MSIT (Ministry of Science and ICT), under the High-Potential Individuals Global Training Program (2021-0-01516) supervised by the IITP (Institute of Information and Communications Technology Planning & Evaluation).

References

[1] *Information Technology-security Techniques-security Requirements for Cryptographic Modules, KS X ISO/IEC 19790*, Korean Agency for Techonology and Standards, Korea, 2015.

[2] A. J. Menezes, P. C. Van Oorschot, and S. A. Vanstone, *Handbook of Applied Cryptography*, CRC Press, FL, USA, 1st edition, 1996.

[3] D. R. Stinson, *Cryptography: Theory and Practice*, Chapman and Hall/CRC, USA, 3rd edition, 2005.

[4] H. Choi, W. Ju, H. Kim, and Y. Yeom, *Guideline for the Collection and Application of Noise Sources on Operating Systems*, TTAS.KO-12.0235/R2, Telecommunication Technology Association, Korea, 2020.

[5] H. Kim, W. Ju, H. Choi, and Y. Yeom, *Guideline for Testing noise Sources used in Software Cryptographic Modules*, TTAK.KO-12.0341, Elecommunication Technology Association, Korea, 2018.

[6] M. Turan, E. Barker, J. Kelsey, M. L. Baish, and M. Boyle, *Recommendation for the Entropy Sources used for Random Bit Generation*, NIST SP800-90B, National Institute of Standards and Technology, USA, 2018.

[7] E. Barker, *Recommendation for Key Management*, NIST SP800-57, National Institute of Standards and Technology, USA, 2020.

[8] E. Barker and J. Kelsey, *Recommendation for Random Bit Generator (RBG) Constructions*, NIST SP800-90C, National Institute of Standards and Technology, USA, 2016.

[9] Z. Gutterman, B. Pinkas, and T. Reinman, "Analysis of the Linux random number generator," IEEE S&P 2006, Berkeley, California, USA, 2006.

[10] M. Vanhoef and F. Piessens, "Predicting, decrypting, and abusing WPA2/802.11 group keys," 25th USENIX Security Symposium, Austin, TX, USA, 2016.

[11] M. P. Pawlowski, A. Jaraand, and M. Ogorzalek, "Harvesting entropy for random number generation for Internet of things constrained devices using on-board sensors," *MDPI, Sensors*, vol. 15, no. 10, pp. 26838–26865, 2015.

[12] G. Souaki and K. Halim, *Random number generation based on MCU sources for IoT application*, ATSIP'2017, Fez, Morroco, 2017.

[13] K. Wallace, K. Moran, E. Novak, G. Zhou, and K. Sun, "Toward sensor-based random number generation for mobile and IoT devices," *IEEE Internet of Things Journal*, vol. 3, no. 6, pp. 1189–1201, 2016.

[14] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, and P. A. Manzagol, "Stacked denoising autoencoders: learning useful representations in a deep network with a local denoising criterion," *Journal of Machine Learning Research*, vol. 11, pp. 3371–3408, 2010.

[15] G. E. Hinton, S. Osindero, and Y. W. Teh, "A fast learning algorithm for deep belief nets," *Neural Computation*, vol. 18, no. 7, pp. 1527–1554, 2006.

[16] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[17] T. Mikolov, S. Kombrink, L. Burget, J. Cernocky, and S. Khudanpur, *Extensions of recurrent neural network language model*, IEEE ICASSP, 2011.

[18] C. Dong, C. Loy, K. He, and X. Tang, "Image super-resolution using deep convolutional networks," *IEEE TPAMI*, vol. 38, no. 2, pp. 295–307, 2016.

[19] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," *Proceedings of the IEEE CVPR*, pp. , 20153431–3440, 2015.

[20] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in Neural Information Processing Systems*, pp. 1097–1105, 2012.

[21] M. Lin, Q. Chen, and S. Yan, "Network in network," (2013), https://arxiv.org/abs/1312.4400.

[22] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," (2014), https://arxiv.org/abs/1409.1556.

[23] C. Szegedy, W. Liu, Y. Jia et al., "Going deeper with convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1–9, 2015.

[24] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," (2015), https://arxiv.org/abs/1512.03385.

[25] S. Mun, S. Jang, J. H. Lee, and J. S. Lee, "Machine learning and deep learning technology trends," *Korean Institute of Communications and Information Sciences*, vol. 33, no. 10, pp. 49–56, 2016.

[26] H. Park, *A study on the security evaluation for cryptographically secure random number generators, [Ph.D. thesis]*, Dept. Financial Information Security, Kookmin Univ, Seoul, KR, 2020.

[27] J. Yang, S. Zhu, T. Chen, Y. Ma, N. Lv, and J. Lin, "Neural network based min-entropy estimation for random number generators," *Security and Privacy in Communication Networks*, *SecureComm*, 2018Springer, 2018.

[28] A statistical test suite for random and pseudorandom number generators for cryptographic applications, NIST SP800-22. (2010).

[29] W. Killmann and W. Schindler, "A proposal for functionally classes and evaluation methodology for true (physical) random number generators," *AIS*, 2001.

[30] L. Dorrendorf, Z. Gutterman, and B. Pinkas, "Cryptanalysis of the random number generator of the Windows operating system," *ACM Transactions on Information and System Security*, vol. 13, no. 1, pp. 1–32, 2009.

[31] "CryptGenRandom function (wincrypt.h)," Microsoft (2021), http://docs.microsoft.com/en-us/windows/win32/api/wincrypt/.

[32] A. Host, "Number of Internet of things (IoT) connected devices worldwide from 2019 to 2030Statista(2021), http://statista.com/statistics/1183457/iot-connected-devices-worldwide/.

[33] S. Higginbotham, "IoT news of the weekStacey on IoT(2021), http://staceyoniot.com/iot-news-of-the-week-for-sept-10-2021/.

[34] Z. Cai and Z. He, "Trading private range counting over big IoT data," in *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*, pp. 144–153, Dallas, TX, USA, 2019.

[35] X. Zheng and Z. Cai, "Privacy-preserved data sharing towards multiple parties in industrial IoTs," *IEEE Journal on Selected Areas in Communications (JSAC).*, vol. 38, no. 5, pp. 968–979, 2020.

[36] Z. Cai, Z. He, X. Guan, and Y. Li, "Collective data-sanitization for preventing sensitive information inference attacks in social networks," *IEEE Transactions on Dependable and Secure Computing*, vol. 15, no. 4, pp. 577–590, 2018.

[37] Z. Cai and X. Zheng, "A private and efficient mechanism for data uploading in smart cyber-physical systems," *IEEE Transactions on Network Science and Engineering (TNSE).*, vol. 7, no. 2, pp. 766–775, 2020.

[38] P. C. Kocher, "Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems," *Advances in Cryptology*, vol. 1109, pp. 104–113, 1996.

[39] A. Golder, D. Das, J. Danial, S. Ghosh, S. Sen, and A. Raychowdhury, "Practical approaches toward deep-learning-based cross-device power side-channel attack," *IEEE Transactions on VLSI Systems*, vol. 27, no. 12, pp. 2720–2733, 2019.

[40] Language reference. Arduino, https://www.arduino.cc/reference/en/".

[41] Advanced Encryption Standard (AES), Federal Information Processing Standards Publication 197, (2001).