telecom

MDPI

Article

Symmetric Keys for Lightweight Encryption Algorithms Using a Pre–Trained VGG16 Model

Ala'a Talib Khudhair *, Abeer Tariq Maolood and Ekhlas Khalaf Gbashi

Computer Science Department, University of Technology, Baghdad 10066, Iraq; abeer.t.maolood@uotechnology.edu.iq (A.T.M.); ekhlas.k.gbashi@uotechnology.edu.iq (E.K.G.)
* Correspondence: cs.22.20@grad.uotechnology.edu.iq

Abstract: The main challenge within lightweight cryptographic symmetric key systems is striking a delicate balance between security and efficiency. Consequently, the key issue revolves around crafting symmetric key schemes that are both lightweight and robust enough to safeguard resource-constrained environments. This paper presents a new method of making long symmetric keys for lightweight algorithms. A pre–trained convolutional neural network (CNN) model called visual geometry group 16 (VGG16) is used to take features from two images, turn them into binary strings, make the two strings equal by cutting them down to the length of the shorter string, and then use XOR to make a symmetric key from the binary strings from the two images. The key length depends on the number of features in the two images. Compared to other lightweight algorithms, we found that this method greatly decreases the time required to generate a symmetric key and improves defense against brute force attacks by creating exceptionally long keys. The method successfully passed all 15 tests when evaluated using the NIST SP 800-22 statistical test suite and all Basic Five Statistical Tests. To the best of our knowledge, this is the first research to explore the generation of a symmetric encryption key using a pre–trained VGG16 model.

Keywords: lightweight cryptographic; transfer learning; pre–trained VGG16; feature extraction

check for updates

Citation: Khudhair, A.T.; Maolood, A.T.; Gbashi, E.K. Symmetric Keys for Lightweight Encryption Algorithms Using a Pre–Trained VGG16 Model. *Telecom* **2024**, *5*, 892–906. https://doi.org/10.3390/telecom5030044

Academic Editor: Sotirios K. Goudos

Received: 27 July 2024
Revised: 19 August 2024
Accepted: 27 August 2024
Published: 3 September 2024

Copyright: © 2024 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

The demand for lightweight encryption techniques in the field of information security has significantly increased due to the widespread usage of IoT devices, embedded systems, and mobile applications [1]. These systems often have limited computational resources, requiring encryption methods thata offer strong security without causing excessive computational load. Generating cryptographic keys plays a vital role in the security and efficiency of lightweight encryption algorithms. There are only a few studies that look at generating symmetric keys, such as [2] Khudhair, A. T. and Maolood A. T. propose generating a new strong key for AES encryption method depending on 2D Henon Map, and [3] Jamil, A. S; Azeez R. A.; Hassan N. F. propose using image feature extraction to generate a key for encryption in cyber security medical environments. In [4] Khudhair, A. T. and Maolood A. T. propose generating robust key based on neural networks and chaos theory. The keys generated by the above methods are short, so they are vulnerable to brute force attacks.

Recent years have seen extensive use of deep learning in classification and recognition tasks [5]. Pre–trained VGG16, especially those trained on large-scale image datasets like ImageNet, have shown a remarkable ability to capture detailed and discriminative features from images [6]. This research will demonstrate the use of the pre–trained VGG16 model for creating lengthy symmetric keys, highlighting the uniqueness of our approach as there are no direct precedents in this field.

This study shows a new way to make long symmetric keys for lightweight cryptography. It does this by using a pre–trained VGG16 to extract features from two images, processing the image features to make binary strings, equalizing the two binary strings by

cutting them down to the length of the shorter one and using an XOR operation between the binary strings from the two images to make a symmetric key. This paper highlights the following key contributions:

a. Introduce a new method that leverages the pre–trained VGG16 approach for generating lengthy symmetric keys. Reduce the number of keys required for long inputs, resulting in a shorter time for key generation.
b. The key length depends on the number of features in the two images.
c. The proposed methodology offers flexibility in key generation by not restricting it to a specific type of image.
d. Incorporating feature extraction into key generation increases randomness and unpredictability.
e. By creating exceptionally long keys, it improves defenses against brute force attacks.

The remaining parts of this paper are organized as follows: Section 2 introduces the theoretical background. Section 3 illustrates the proposed method and provides a simple example of key generation. Section 4 discusses randomness testing. Section 5 covers symmetric key length. Section 6 discusses the complexity of time. Section 7 examines brute force attacks. Finally, Section 8 presents conclusions and outlines future directions for research.

2. Theoretical Background

2.1. Symmetric Key

Symmetric key cryptography employs the same key (K) for encryption and decryption, as illustrated in Figure 1. The sender encrypts the plaintext using key 'K' and transmits the resulting ciphertext to the receiver. Subsequently, the receiver decrypts the cipher data using the identical key 'K', restoring it to its original form [7,8]. Maintaining the secrecy of the key is crucial, ensuring its exclusive exchange through a secure channel between the sender and the receiver [9]. There are many symmetric encryption algorithms, such as:

Figure 1. Symmetric key encryption.

2.1.1. Advanced Encryption Standard (AES)

To address the need for enhanced protection and performance, NIST initiated a call for cipher candidates in 1997 to establish a new encryption standard. This led to the replacement of the existing DES and 3DES encryption algorithms with the new AES encryption algorithms. The Feistel structure of the AES symmetric block cipher allows it to handle a 128-bit block size and supports three key lengths—128, 192, and 256 bits—enabling 10, 12, and 14 rounds of encryption and decryption with the same key. The vector design of Rijndael, which forms the basis of the AES, provides significant security, and its scalability of up to 256 bits offers resistance to potential attacks [7,10]. Figure 2 illustrates the AES algorithm.

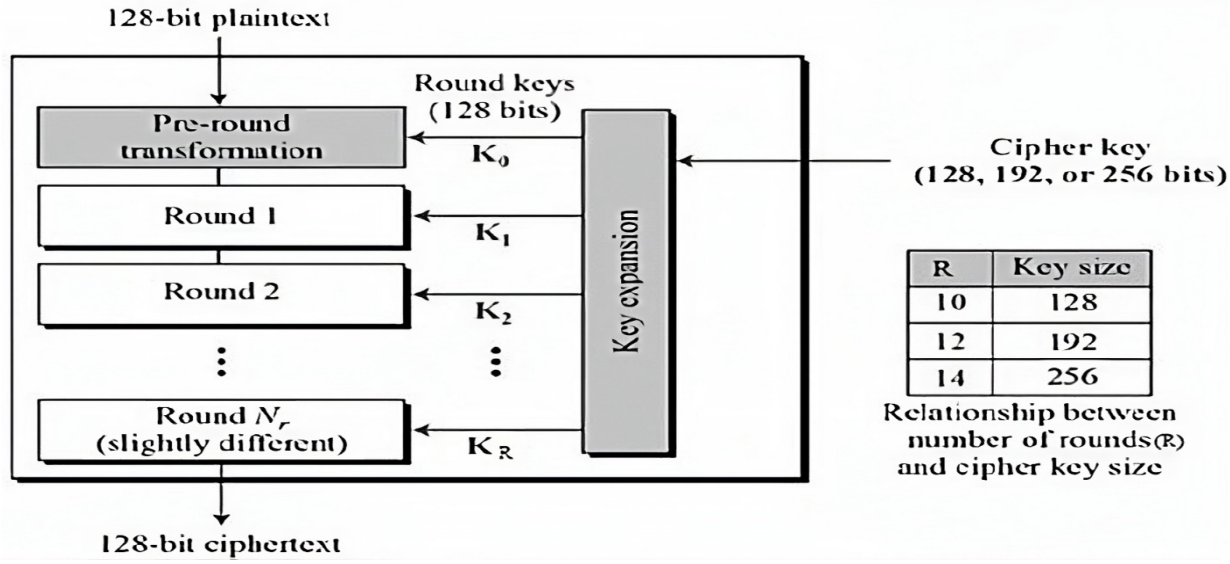


Figure 2. AES encryption algorithm.

2.1.2. Data Encryption Standard (DES)

DES, is an early symmetric encryption method developed by IBM in 1972. This symmetric key encryption technology is used to secure electronic data. DES operates as a block cipher with a 64-bit key, although only 56 bits are used effectively while the remaining bits are for parity. The encryption process involves 16 rounds of circular permutations, along with initial and final permutations. In typical threat environments, the 56-bit key size, which generates 7.2×10^{16} possible keys, provides the cryptographic strength of DES [7,11], as illustrated in Figure 3.

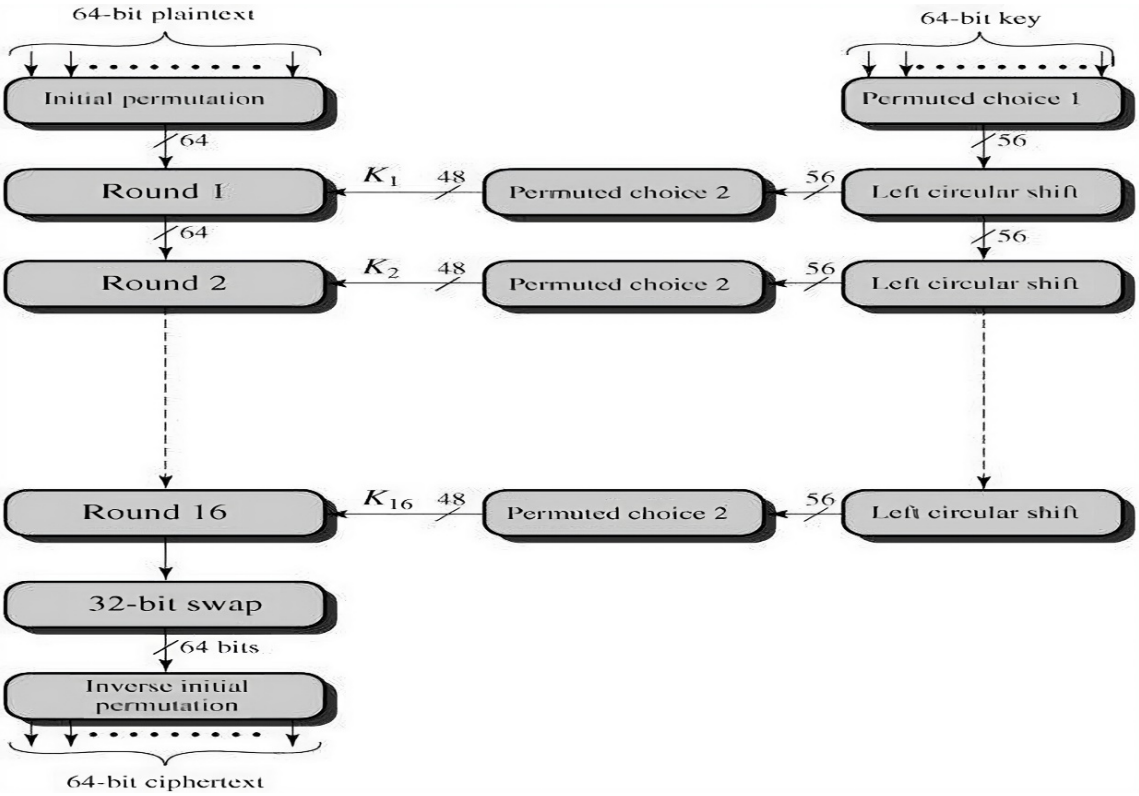


Figure 3. DES encryption algorithm.

2.1.3. Blowfish

Blowfish is a symmetric cipher with a variable key length based on a Feistel structure. It features a 64-bit block size and supports key lengths ranging from 32 to 448 bits. The algorithm employs 16 rounds and utilizes a wide S-box that depends on the key. Blowfish uses four S-boxes, and the same algorithm is used for decryption in reverse [7,12], as illustrated in Figure 4.

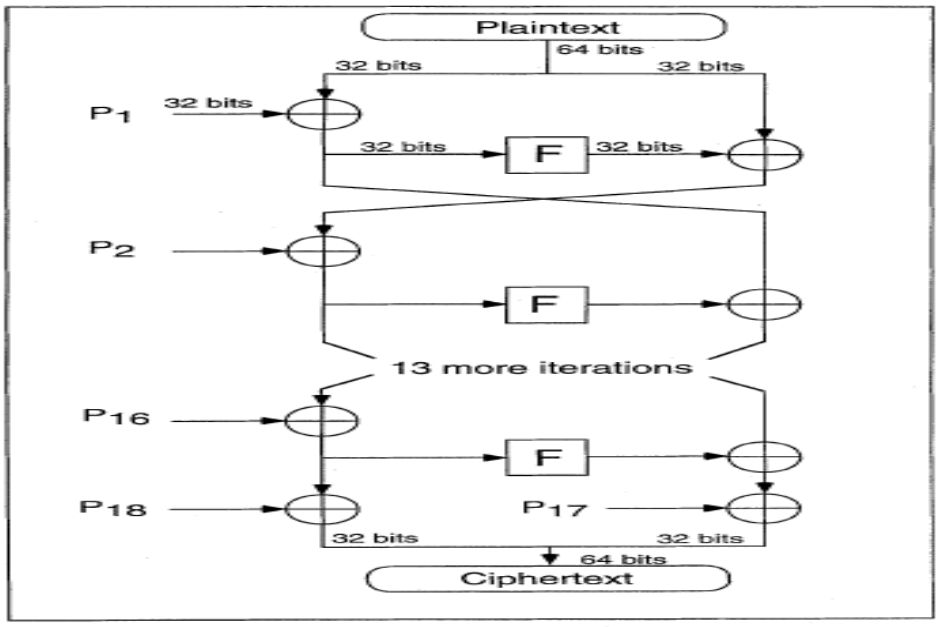


Figure 4. Blowfish encryption algorithm.

2.2. Transfer Learning

Transfer learning in machine learning involves reusing a pre–trained model to tackle a new problem [13]. This technique offers several advantages, including reduced training time, improved neural network performance in most cases, and the ability to work with limited data [14]. Among the most popular pre–trained models are VGG-16, VGG-19, Inception V3, Xception, and ResNet-50 [15]. Figure 5 illustrates the contrast between conventional machine learning and transfer learning processes. In conventional machine learning, each distinct task is learned separately using different learning systems. In contrast, transfer learning extracts knowledge from previous source tasks and applies it to target tasks.

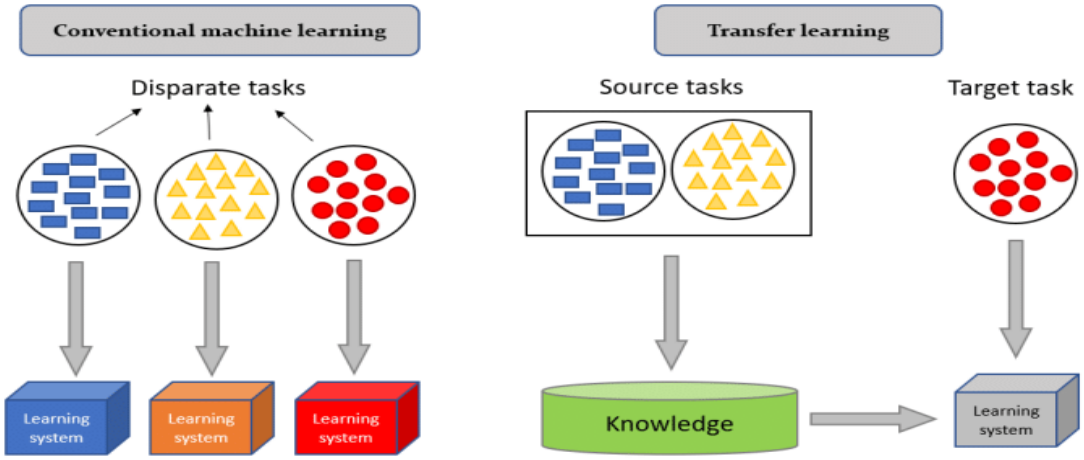


Figure 5. Conventional Machine Learning vs. Transfer Learning.

2.3. Feature Extraction Using VGG16 Architecture

The visual geometry group (VGG) at the University of Oxford developed the VGG-16 model, which stands out for its depth and consists of 16 layers. Figure 6 illustrates how we can use the VGG-16 model for feature extraction, with 13 convolutional layers and 3 fully connected layers [16]. Each block, comprising multiple convolutional layers and a max-pooling layer to facilitate down–sampling, structures the layers [17]. Known for its simplicity and effectiveness, VGG-16 demonstrates strong performance across various computer vision tasks [18].

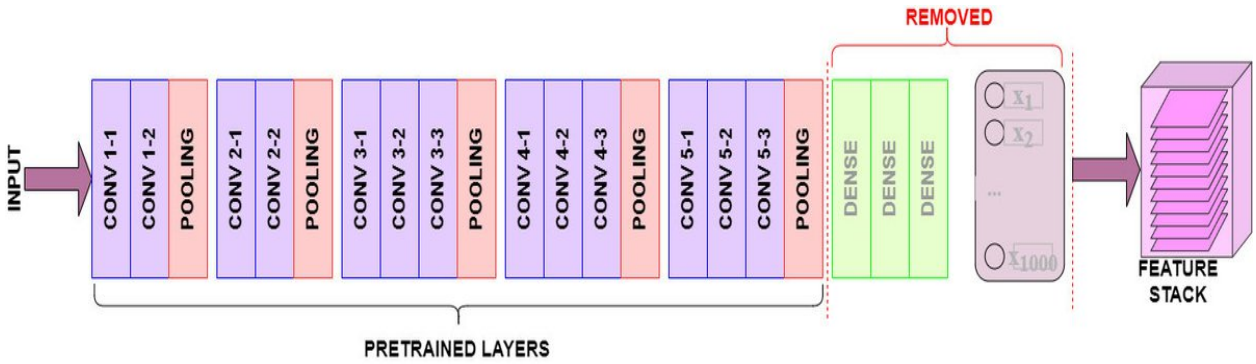


Figure 6. The graphic above depicts the "top" portion of the model being removed. A 3D stack of feature maps convolves the remaining pre–trained output layers.

The VGG-16 network structure [19], illustrated in Figure 6, can be described as follows:

- ✓ The initial two convolutional layers contain 64 feature kernel filters, each sized at 3×3. Upon passing an RGB image input (with a depth of 3) through these layers, the dimensions transform to $224 \times 224 \times 64$. Subsequently, the output undergoes a max pooling layer with a stride of 2.
- ✓ Following are the third and fourth convolutional layers with 128 feature kernel filters, each also sized at 3×3. These layers are succeeded by a max pooling layer with a stride of 2, resulting in output dimensions of $56 \times 56 \times 128$.
- ✓ The fifth, sixth, and seventh layers are convolutional layers with 256 feature maps, each with a kernel size of 3×3. These layers are followed by a max pooling layer with a stride of 2.
- ✓ The eighth through thirteenth layers are two sets of convolutional layers with 512 kernel filters, each with a size of 3×3. These layers are followed by a max pooling layer with a stride of 1.
- ✓ The fourteenth and fifteenth layers are fully connected hidden layers with 4096 units each, followed by a softmax output layer (the sixteenth layer) with 1000 units.

a. Convolution layer: The convolutional layer process entails using a kernel matrix to generate a feature map from the input matrix, achieved through the mathematical operation of convolution [20]. This linear operation involves sliding the kernel matrix across the input matrix, performing element-wise matrix multiplication at each position, and accumulating the results onto the feature map, as shown in Figure 7 [21]. Convolution is widely utilized in various fields like image processing, statistics, and physics [22]. The calculation of the convoluted image follows this procedure:

$$S\,(i,\,j) = \sum_x \sum_y I\,(\mathrm{x},\,\mathrm{y})\,k\,(i\,-\,\mathrm{x},\,j\,-\,\mathrm{y}) \tag{1}$$

where I is a 2-dimensional image input, k is a 2-dimensional kernel filter, and S is a 2-dimensional output feature map.

b. Non-linear activation functions, such as the Rectified Linear Unit (ReLU), serve as nodes in a neural network that follow the convolutional layer. These functions are responsible for introducing nonlinearity to the input signal through a piecewise

linear transformation. In the case of ReLU, the function outputs the input if it is positive and outputs zero if it is negative, as shown in Figure 8 [23].

c. Pooling layer: The convolutional layer's feature map output is limited because it preserves the exact location of features in the input, making it sensitive to minor changes like cropping or rotation [24]. To address this issue, we can introduce down–sampling in the convolutional layers by adding a pooling layer after the nonlinearity layer [25]. This pooling helps to make the representation insensitive to small translations of the input, ensuring that most of the pooled outputs remain unchanged even with slight adjustments to the input [26]. The max-pooling unit extracts the maximum value from a set of four values, as illustrated in Figure 9. Average pooling is an alternative option for pooling layers [27].

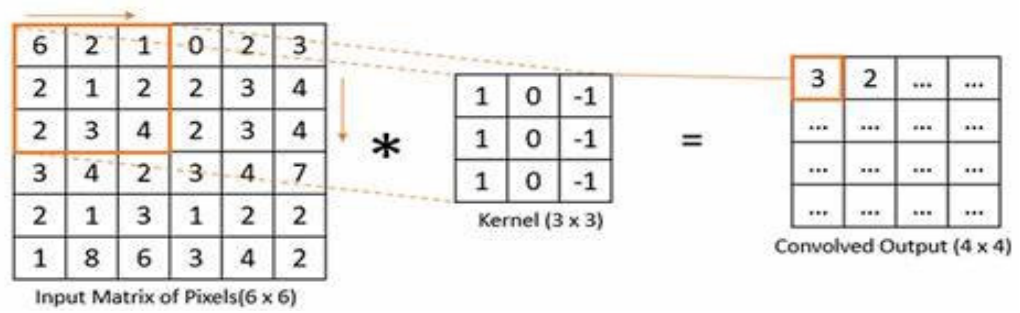


Figure 7. The results of matrix multiplication are summed onto the feature map.

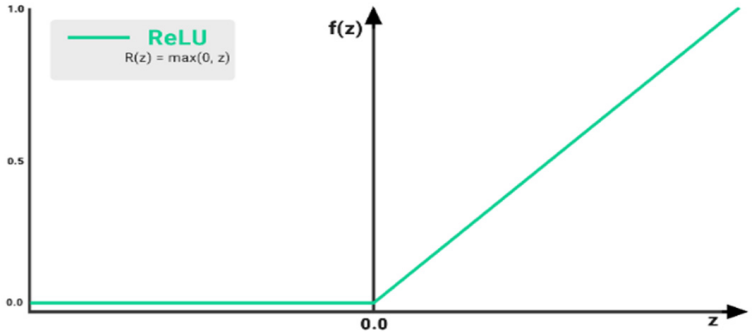


Figure 8. Activation function (ReLU).

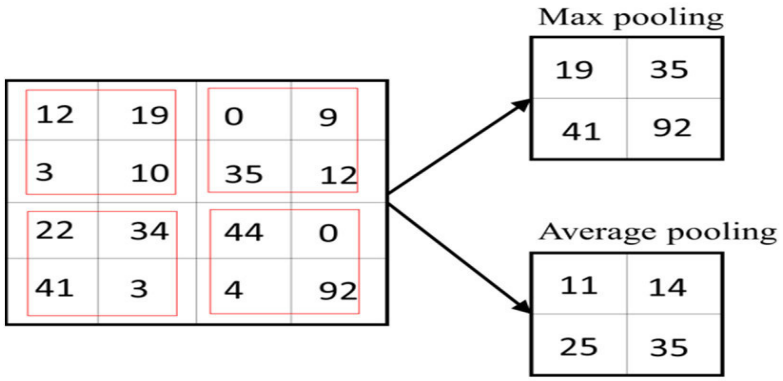


Figure 9. Types of pooling.

3. The Proposed Method

This paper creates a long symmetric key using a deep CNN model based on the VGG16 architecture, which is pre–trained on the ImageNet dataset with 1000 classes. We implement the model using the Tensorflow backend of the Keras library. Each image generates features with dimensions of 7×7 and 512 channels, resulting in $7 \times 7 \times 512 = 25{,}088$ features. A

flattening layer transforms these features into a one-dimensional vector. Algorithm 1 and Figure 10 detail the processing of the extracted features to generate the long symmetric key.

Algorithm 1. Symmetric Key Generation Using Pre–trained VGG16

Input: Two images (randomly)
Output: Symmetric key (binary)
Begin

1. Load the pre–trained VGG16 model.
2. Load the first image.
3. Preprocess the image by resizing it to 224 × 224 pixels. //The required input size for VGG16
4. Extract features from the image.
5. Remove all instances of 0.0 from the extracted features.
6. Remove decimal points from all the remaining features.
7. Convert each resulting feature from step 6 into a binary string.
8. Load the second image.
9. Repeat steps 3–7 for the second image.
10. To equalize the two binary strings, truncate them to the length of the shorter one.
11. Perform an XOR operation between the first image's binary string and the second image's binary string.

End

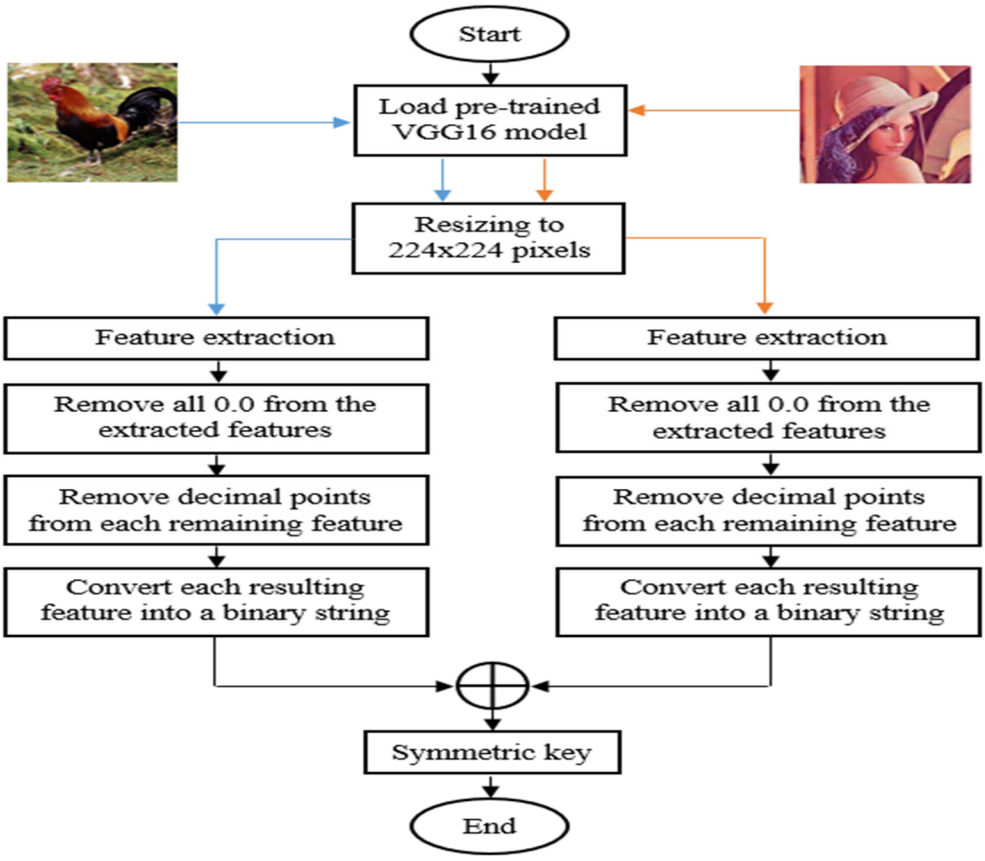


Figure 10. Flowchart of the proposed symmetric key generation using pre–trained VGG16 model.

3.1. Example

Below is a simple example of generating long symmetric keys using the pre–trained VGG16.

- The user randomly chooses the first image

- The user randomly chooses the second image

- The features are extracted from the first image. The result appears as a 3D array since the image is in color, as shown below.

[[[[0.0, 0.0, 0.0, 0.0, 6.903665065765381, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 34.10573196411133, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 67.41827392578125, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 31.747112274169922, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 15.003043174743652, 0.0, 0.0, 0.0, 0.0, 0.0, 17.883718490600586, 0.0, 0.0, 31.02227210998535, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 3.426398515701294, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 4.979990482330322, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 67.9732666015625, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 2.141775369644165, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 17.164846420288086, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.9679595232009888, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 11.956241607666016, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 2.2587838172912598, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 6.765922546386719, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 19.34865951538086, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 5.301035404205322, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 10.16125774383545, 0.0, 0.0, 0.0, 17.666433334350586, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 6.217296600341797, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 35.38439178466797, 0.0, 0.0, 0.0, 0.0, 0.0,. . .]]]]

- The features are extracted from the second image. The result appears as a 3D array since the image is in color, as shown below.

[[[[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 24.223085403442383, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 10.136581420898438, 9.996827125549316, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 43.26572036743164, 0.0, 0.0, 9.374469757080078, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 4.680146217346191, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 11.460838317871094, 8.864582061767578, 0.0, 0.0, 12.735118865966797, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.2287561893463135, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 6.5052361488342285, 0.0, 0.0, 0.0, 0.0, 4.149342060089111, 7.862621784210205, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 37.46705627441406, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 3.7152833938598633, 0.0, 0.0, 11.048995018005371, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 13.8164701461792, 1.718098759651184, 0.0, 11.642448425292969, 0.0, 0.0, 0.0, 0.0, 5.835142612457275, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 3.547330379486084, 8.25907039642334, 0.0, 3.026193141937256, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 5.3880109786987305, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.9568023681640625, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 3.869873046875, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.10132180899381638, 0.0, 0.0, 0.0, 0.0, 9.61951732635498, 29.360055923461914, . . .]]]]

- We eliminated all instances of 0.0 from the extracted features in the first image. The first image contains 2832 features, as shown below.

[6.903665065765381, 34.10573196411133, 67.41827392578125, 31.747112274169922, 15.003043174743652, 17.883718490600586, 31.02227210998535, 3.426398515701294, 4.979990482330322, 67.9732666015625, 2.141775369644165, 17.164846420288086, 0.9679595232009888, 11.956241607666016, 2.2587838172912598, 6.765922546386719, 19.34865951538086, 5.301035404205322, 10.16125774383545, 17.666433334350586, 6.217296600341797, 35.38439178466797, 12.481855392456055, 49.49517822265625, 2.6517748832702637, 58.775062561035156, 14.709327697753906, 4.851640224456787, 32.54026794433594, 18.19671058654785, 38.65156936645508, 62.830291748046875, . . .]

- We eliminated all instances of 0.0 from the extracted features in the second image. The second image contains 3644 features, as shown below.

[24.223085403442383, 10.136581420898438, 9.996827125549316, 43.26572036743164, 9.374469757080078, 4.680146217346191, 11.460838317871094, 8.864582061767578, 12.735118865966797, 1.2287561893463135, 6.5052361488342285, 4.149342060089111, 7.862621784210205, 37.46705627441406, 3.7152833938598633, 11.048995018005371, 13.8164701461792, 1.718098759651184, 11.642448425292969, 5.835142612457275, 3.547330379486084, 8.25907039642334, 3.026193141937256, 5.3880109786987305, 1.9568023681640625, 3.869873046875, 0.10132180899381638, 9.61951732635498, 29.360055923461914, 11.005602836608887, 3.671891927719116, 25.473989486694336, 9.130292892456055, 7.472241401672363, 11.316661834716797, 8.249753952026367, 15.941848754882812, 6.701721668243408, 2.259157180786133, 33.08333969116211, . . .]

- Remove decimal points from all the resulting features for the first image, then convert them into a binary string. The output is a binary string of 150,728 bits.

Feature	**Remove the Dot (.)**	**Convert to Binary**
6.903665065765381	6903665065765381	11000100001101101100011000111111001001010101000000101
34.10573196411133	3410573196411133	11000001110111100101111100111101101001000000111111101
67.41827392578125	6741827392578125	10111111100111010100000000010110111110010001001001101
31.747112274169922	31747112274169922	11100001100100111010100100101011011111100010000010000010
15.003043174743652	15003043174743652	11010101001101001100000011001011111010111111001100100
. . .	. . .	. . .

- Remove decimal points from all the resulting features for the second image, then convert them into a binary string. The output is a binary string of 193,752 bits.

Feature	**Remove the Dot (.)**	**Convert to Binary**
24.223085403442383	24223085403442383	10101100000111011000100100100001101011000001100110011111
10.136581420898438	10136581420898438	10010000000011001010101100011011010100010110001000011 0
9.996827125549316	9996827125549316	10001110000100000011111011000110110011000010010000010 0
43.26572036743164	4326572036743164	11110101111011111110100011010101100001011111111111100
9.374469757080078	9374469757080078	10000101001110000001111101001111010100000100100000111 0
. . .	. . .	. . .

- To equalize the two binary strings, truncate them to the length of the shorter one and perform an XOR operation between the binary string of the first image and the binary string of the second image. The output is a binary string of 150,728 bits, as shown below.

[01101000001010110100111100011110100010010100100110110001001011101110101110101111
11101101110000010101110101110101110111110001000011111110101001010111010000001011100
01100011110011010110011011011101001000001111011110111110100011110101100011100011000
00111111111101011100111010111110110010110110100010101100000010010011011001101011000
01001110000111110001010111010011101010111110010001110110111010111000110111111011101
10111011001011100100110110001011001001001100110011100100111111100100111101111111100
10100100100011000010111011111101011110111001000111001010000011000001000110000111 00

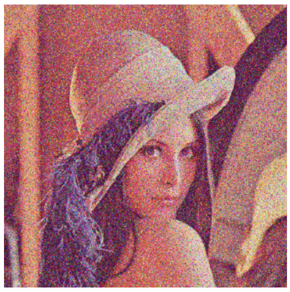
01011000011111011010111101111000010101111011011010111001100111100011110011110111100000110011011011010000110011111100101001010010000100101000000010111010101001010101001101111110011111011110110111010100000101000010000001110001111010001010111101000111001111001110110001010010100101101001000011101101000100010010010101110101011100110101110010110010111101011000110011110001101011100010110001000100001110110100010110101000100111111001110101111110111001111000000001101010101001100111110110110001011011111010110110100101001111011100101000000001011000000001010100100001010001111101110000110011111011010, . . .]

3.2. The Sensitivity of the Symmetric Key to Any Alterations Made to One of the Reference Images

An effective method should demonstrate a high degree of sensitivity to even the slightest changes in the base image. For instance, minor modifications such as altering a single bit should lead to substantial differences in the features extracted from the image. These significant changes in the extracted features should, in turn, result in the generation of completely different keys. Here, a simple example shows how changing a single bit in the base image completely changes the features extracted from it.

Base image

single bit in the base image is changed



- The features are extracted from the base image. The result appears as a 3D array since the image is in color, as shown below.

[[[[0.0, 0.0, 0.0, 0.0, 6.903665065765381, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 34.10573196411133, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 67.41827392578125, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 31.747112274169922, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 15.003043174743652, 0.0, 0.0, 0.0, 0.0, 0.0, 17.883718490600586, 0.0, 0.0, 31.02227210998535, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 3.426398515701294, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 4.979990482330322, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 67.9732666015625, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 2.141775369644165, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 17.164846420288086, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.9679595232009888, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 11.956241607666016, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 2.2587838172912598, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 6.765922546386719, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 19.34865951538086, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 5.301035404205322, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 10.16125774383545, 0.0, 0.0, 0.0, 17.666433334350586, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 6.217296600341797, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 35.38439178466797, 0.0, 0.0, 0.0, 0.0, 0.0,. . .]]]]

- The features are extracted from the base image after a single bit is changed. The result appears as a 3D array since the image is in color, as shown below.

[[[[0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 8.439230918884277, 0.0, 12.917805671691895, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,

0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 2.9626998901367188, 0.0, 0.0, 43.08259963989258, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 3.9462945461273193, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 48.83059310913086, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 12.243735313415527, 27.153099060058594, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,3.810386896133423, 37.272727966308594, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 5.046123504638672, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 26.434804916381836, 2.660520076751709, 0.0, 0.0,7.4828314781188965, 0.0, 0.0, 1.8224953413009644, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 23.51656723022461, 0.0, 6.485126495361328, 0.0, 0.0, 0.0, 0.0, 13.915960311889648, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 6.456112861633301, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 27.75977325439453, 0.0, 0.0, 0.0, 0.0, 14.999650955200195, 0.5237445831298828, 0.0, 0.0, 0.0, 0.0, 0.0, 4.312615871429443, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 28.290889739990234, 0.0, 0.0, 0.0, 1.773626446723938, 22.507610321044922, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 4.360454559326172, 0.0, 7.555282115936279, 0.0, 0.0, 0.0, 55.96969985961914, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 8.2135591506958, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 6.036996841430664, 0.0, 8.161231994628906, 0.0, 3.563197612762451, 0.0, 0.0, 0.0, 0.0, 34.11822509765625, 0.0, 0.0,. . .]]]]

4. Randomness Testing

We performed a comprehensive set of tests to confirm the accuracy of our proposed method. This testing process involved two main components: Initially, we examined the key generator with the Basic Five Statistical Tests, and then we assessed its performance using the NIST SP 800-22 statistical test suite.

4.1. Basic Five Statistical Tests

We evaluated the proposed Algorithm 1 for randomness using the Basic Five Statistical Tests. This assessment validated the randomness of the symmetric keys generated. Table 1 presents the pass value from the Basic Five Statistical Tests. This suite includes tests for the frequency, run, poker, serial, and autocorrelation of a given input. We generated 18,750,723 bits from 290 different images and tested them in the suite. Table 1 demonstrates that the generated symmetric keys are highly unpredictable and random, making it challenging for adversaries to guess the keys.

Table 1. Pass values of Algorithm 1 for 18,750,723 bits in Basic Five Statistical Tests.

Tests		Freedom Degree	18,750,723 Bits
Frequency Test		Must be <= 3.84	Pass = 0.490
Run Test	T0	Must be <= 32.386	Pass = 23.280
	T1	Must be <= 31.126	Pass = 30.734
Poker Test		Must be <= 11.1	Pass = 2.871
Serial Test		Must be <= 7.81	Pass = 2.524
Auto Correlation Test	Shift No. 1	Must be <= 3.84	Pass = 0.559
	Shift No. 2		Pass = 1.294
	Shift No. 3		Pass = 1.023
	Shift No. 4		Pass = 2.721
	Shift No. 5		Pass = 1.906
	Shift No. 6		Pass = 1.907
	Shift No. 7		Pass = 0.457
	Shift No. 8		Pass = 3.536

Table 1. *Cont.*

Tests		Freedom Degree	18,750,723 Bits
Auto Correlation Test	Shift No. 9	Must be <= 3.84	Pass = 1.010
	Shift No. 10		Pass = 0.054

4.2. NIST SP 800-22 Statistical Test Suite

Algorithm 1 was randomly evaluated using the NIST SP 800-22 statistical test suite, confirming the randomness of the generated symmetric keys. Table 2 shows the P-value and pass rate from the suite's various randomness tests. Tests for frequency, block frequency, cumulative sums, runs, longest runs, rank, FFT, non-overlapping template, overlapping template, universal, approximate entropy, random excursions, random excursions variant, serial, and linear complexity were all included in the evaluation process. We generated and tested a total of 18,750,723 random bits from 290 different images. We evaluated the streams of 30 bits, 41 bits, and 43 bits using the default configuration of the NIST SP 800-22 suite. The results in Table 2 indicate that the symmetric keys generated are highly unpredictable and random, making it difficult for adversaries to guess them.

Table 2. P-values and success rates of Algorithm 1 for 30, 41, and 43 bits in NIST SP 800-22.

Test Name	Assess: 387,840		Assess: 387,840		Assess: 435,671	
	Number of Bits: 30 Bits		Number of Bits: 41 Bits		Number of Bits: 43 Bits	
	p-Value	Pass Rate	*p*-Value	Pass Rate	*p*-Value	Pass Rate
Frequency	0.468595	29/30	0.559523	40/41	0.611108	43/43
Block Frequency	0.671779	30/30	0.460664	41/41	0.460664	43/43
Cumulative Sums	0.804337	30/30	0.811993	41/41	0.927083	42/43
Runs	0.671779	29/30	0.663130	40/41	0.764655	41/43
Longest Run	0.671779	29/30	0.258961	40/41	0.151616	42/43
Rank	0.602458	29/30	0.414525	39/41	0.559523	43/43
Discrete Fourier Transform	0.976060	30/30	0.663130	41/41	0.258961	43/43
Non- Overlapping Templates	0.999896	30/30	0.986869	41/41	0.953553	43/43
Overlapping Templates	0.602458	28/30	0.811993	39/41	0.811993	41/43
Universal	0.602458	30/30	0.227773	41/41	0.811993	43/43
Approximate Entropy	0.534146	30/30	0.460664	41/41	0.509162	43/43
Random Excursions	0.739918	20/20	0.534146	24/24	0.875539	23/23
Random Excursions Variant	0.991468	19/20	0.964295	24/24	0.941144	23/23
Serial	0.253551	30/30	0.764655	41/41	0.559523	43/43
Linear Complexity	0.213309	30/30	0.258961	41/41	0.371101	43/43

The P-value ($\geq$0.01) is crucial for determining randomness and the pass rate [28]. Table 2 displays the P-values, all of which exceed the minimum threshold of 0.01. For the 30-bit, 41-bit, and 43-bit streams, the highest P-values are 0.999896, 0.986869, and 0.953553, respectively. The lowest are 0.213309, 0.227773, and 0.151616, respectively. This proves that Algorithm 1 can produce genuinely random numbers, which are useful for creating symmetric keys in lightweight cryptography.

5. Symmetric Key Length

The key length of a symmetric key is variable depending on the number of features present in the two images, as shown in Table 3.

Table 3. Symmetric key length.

Image 1	Image 2	Key Length in Bits
		106,970 bits
		148,424 bits
		150,728 bits
		157,597 bits

6. Time Complexity

Table 4 juxtaposes the proposed method with well-established cryptographic algorithms like the Advanced Encryption Standard (AES), Data Encryption Standard (DES), Blowfish, Speck, and LBlock algorithms, highlighting key criteria such as key size, time, and fundamental components for key generation. Table 4 mentions the time required to generate the key for the AES, DES, Blowfish, Speck, and LBlock algorithms, as well as the proposed method, which we implemented in Python on a core i7 processor (1165G7 @ 2.80 GHz, 8 GB RAM, 64-bit operating system). Based on the results, we conclude that this method significantly reduces the time needed to generate a symmetric key while simultaneously enhancing protection against brute force attacks by greatly increasing the difficulty of revealing the key. For instance, generating a 128-bit key using the AES algorithm takes approximately 0.053 s. the AES algorithm would require approximately 10 s to generate a 157,597-bit key. However, the proposed method can generate a 157,597-bit key in 0.4171 s, equivalent to generating approximately 1,231 keys of 128 bits each.

Table 4. Comparison of the proposed method with other cryptographic algorithms.

Cipher Name	Key Size (Bits)	Time	Basic Components
AES	128	0.053	Key expansion from a master key
DES	56	0.152	Key scheduling from a master key
Blowfish	32–448	0.054	Key expansion from a master key

Table 4. *Cont.*

Cipher Name	Key Size (Bits)		Time	Basic Components
Speck	64		0.097	Key expansion from a master key
	128		0.130	
LBlock	80		0.180	derived from the original 80-bit master key
Proposed method	Depending on the number of features	106,970	0.3858	Feature extraction by pre–trained VGG16
		148,424	0.4016	
		150,728	0.4018	
		157,597	0.4171	

7. Brute Force Attack

A brute force attack represents a hacking technique that relies on trial and error to decipher encryption keys. These attacks require extensive time and resources, especially when targeting longer keys. A crucial aspect of this method involves generating keys of variable lengths with each iteration. This variability enhances the method's efficacy by thwarting the attacker's ability to anticipate the key's length, thus complicating calculations regarding the number of possible key combinations required for successful decryption. For instance, consider Table 3, wherein keys of lengths 106,970 bits, 148,424 bits, 150,728 bits, and 157,597 bits are generated in a single iteration. Consequently, the number of potential keys expands to 2^{106970}, 2^{148424}, 2^{150728}, and 2^{157597}, respectively, heightening the difficulty of the brute force attack.

8. Conclusions

This paper introduced a new method for generating lengthy symmetric keys for lightweight cryptography, utilizing image features extracted with a pre–trained VGG16 model. This technique effectively merges the robustness of deep learning with the efficiency required for lightweight cryptographic applications. Our method uses VGG16's rich feature representations to make sure that key generation is highly random and unpredictable. This meets important security needs in environments with limited resources. The experimental results show that by creating exceptionally long keys, the proposed method reduces the time required for key generation and improves defenses against brute force attacks. As the first study of its kind, we suggest that subsequent research incorporate this key generation method into lightweight block cipher algorithms.

Author Contributions: Conceptualization, A.T.K.; methodology, A.T.K.; software, A.T.K.; validation, A.T.K., A.T.M. and E.K.G.; formal analysis, A.T.K.; investigation, A.T.K., A.T.M. and E.K.G.; resources, A.T.K.; data curation, A.T.K.; writing—original draft, A.T.K.; writing—review and editing, A.T.K., A.T.M. and E.K.G.; visualization, A.T.K., A.T.M. and E.K.G.; supervision, A.T.M. and E.K.G.; project administration, A.T.K., A.T.M. and E.K.G.; funding acquisition, A.T.K. and A.T.M. All authors have read and agreed to the published version of the manuscript.

Funding: The authors received no specific funding for this study.

Data Availability Statement: Data is contained within the article.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Abd Zaid, M.M.; Hassan, S. Proposal Framework to Lightweight Cryptography Primitives. *Eng. Technol. J.* **2022**, *40*, 516–526. [CrossRef]
2. Khudhair, A.T.; Maolood, A.T. Towards Generating a New Strong key for AES Encryption Method Depending on 2D Henon Map. *Diyala J. Pure Sci.* **2019**, *15*, 53–69. [CrossRef]
3. Jamil, A.S.; Azeez, R.A.; Hassan, N.F. An Image Feature Extraction to Generate a Key for Encryption in Cyber Security Medical Environments. *Int. J. Online Biomed. Eng.* **2023**, *19*, 93–106. [CrossRef]

4. Khudhair, A.T.; Maolood, A.T. Towards Generating Robust Key Based on Neural Networks and Chaos Theory. *Iraqi J. Sci.* **2018**, *59*, 1518–1530. [CrossRef]
5. Siddique, M.F.; Ahmad, Z.; Ullah, N.; Ullah, S.; Kim, J.-M. Pipeline Leak Detection: A Comprehensive Deep Learning Model Using CWT Image Analysis and an Optimized DBN-GA-LSSVM Framework. *Sensors* **2024**, *24*, 4009. [CrossRef]
6. Alawi, A.R.; Hassan, N.F. A Proposal Video Encryption Using Light Stream Algorithm. *Eng. Technol. J.* **2021**, *39*, 184–196. [CrossRef]
7. Stamp, M. *Information Security Principles and Practice*, 2nd ed.; Wiley: Hoboken, NJ, USA, 2011.
8. Abd Aljabar, R.W.; Hassan, N.F. Encryption VoIP based on Generated Biometric Key for RC4 Algorithm. *Eng. Technol. J.* **2021**, *39*, 209–221. [CrossRef]
9. Maolood, A.T.; Gbashi, E.K.; Mahmood, E.S. Novel lightweight video encryption method based on ChaCha20 stream cipher and hybrid chaotic map. *Int. J. Electr. Comput. Eng.* **2022**, *12*, 4988–5000. [CrossRef]
10. Gbashi, E.K.; Maolood, A.T.; Jurn, Y.N. Privacy Security System for Video Data Transmission in Edge-Fog-cloud Environment. *Int. J. Intell. Eng. Syst.* **2023**, *16*, 307–318. [CrossRef]
11. Khudhair, A.T.; Maolood, A.T.; Gbashi, E.K. Symmetry Analysis in Construction Two Dynamic Lightweight S-Boxes Based on the 2D Tinkerbell Map and the 2D Duffing Map. *Symmetry* **2024**, *16*, 872. [CrossRef]
12. Raza, M.S.; Sheikh, M.N.A.; Hwang, I.-S.; Ab-Rahman, M.S. Feature-Selection-Based DDoS Attack Detection Using AI Algorithms. *Telecom* **2024**, *5*, 333–346. [CrossRef]
13. Mewada, H. Extended Deep-Learning Network for Histopathological Image-Based Multiclass Breast Cancer Classification Using Residual Features. *Symmetry* **2024**, *16*, 507. [CrossRef]
14. Keawin, C.; Innok, A.; Uthansakul, P. Optimization of Signal Detection Using Deep CNN in Ultra-Massive MIMO. *Telecom* **2024**, *5*, 280–295. [CrossRef]
15. Sekhar, K.S.R.; Babu, T.R.; Prathibha, G.; Ming, K.V.L.C. Dermoscopic image classification using CNN with Handcrafted features. *J. King Saud Univ.–Sci.* **2021**, *33*, 101550. [CrossRef]
16. Tammina, S. Transfer learning using VGG-16 with Deep Convolutional Neural Network for Classifying Images. *Int. J. Sci. Res. Publ.* **2019**, *9*, 143–150. [CrossRef]
17. Peixoto, J.; Sousa, J.; Carvalho, R.; Santos, G.; Mendes, J.; Cardoso, R.; Reis, A. Development of an Analog Gauge Reading Solution Based on Computer Vision and Deep Learning for an IoT Application. *Telecom* **2022**, *3*, 564–580. [CrossRef]
18. Rana, S.; Mondal, M.R.H.; Parvez, A.H.M.S. A New Key Generation Technique based on Neural Networks for Lightweight Block Ciphers. *Int. J. Adv. Comput. Sci. Appl.* **2021**, *12*, 208–216. [CrossRef]
19. Alshehri, A.; AlSaeed, D. Breast Cancer Diagnosis in Thermography Using Pre-Trained VGG16 with Deep Attention Mechanisms. *Symmetry* **2023**, *15*, 582. [CrossRef]
20. Simonyan, K.; Zisserman, A. Very deep convolutional networks for large-scale image recognition. *arXiv* **2014**, arXiv:1409.1556. [CrossRef]
21. Akhand, M.A.H.; Roy, S.; Siddique, N.; Kamal, M.A.S.; Shimamura, T. Facial Emotion Recognition Using Transfer Learning in the Deep CNN. *Electronics* **2021**, *10*, 1036. [CrossRef]
22. Hashem, M.I.; Kuban, K.H. Key generation method from fingerprint image based on deep convolutional neural network model. *Nexo Rev. Cient.* **2023**, *36*, 906–925. [CrossRef]
23. Jumaah, M.A.; Ali, Y.H.; Rashid, T.A.; Vimal, S. FOXANN: A Method for Boosting Neural Network Performance. *Journal of Soft Comput. Comput. Appl.* **2024**, *1*, 1001.
24. Erkan, U.; Toktas, A.; Enginoğlu, S.; Enver Akbacak, E.; Dang, N.H.T. An image encryption scheme based on chaotic logarithmic map and key generation using deep CNN. *Multimed. Tools Appl.* **2022**, *81*, 7365–7391. [CrossRef]
25. Patgiri, R. symKrypt: A Lightweight Symmetric-Key Cryptography for Diverse Applications. *Comput. Inf. Sci.* **2022**, *1055*, 1–30. [CrossRef]
26. Socasi, F.Q.; Vera, L.Z.; Chang, O. A Deep Learning Approach for Symmetric-Key Cryptography System. In Proceedings of the Future Technologies Conference, Xi'an, China, 24–25 October 2020; pp. 539–552. [CrossRef]
27. Wang, Y.; Bing, L.; Zhang, Y.; Jiaxin, W.; Qianya, M. A Secure Biometric Key Generation Mechanism via Deep Learning and Its Application. *Appl. Sci.* **2021**, *11*, 8497. [CrossRef]
28. Kuznetsov, O.; Zakharov, D.; Frontoni, E. Deep learning-based biometric cryptographic key generation with post-quantum security. *Multimed. Tools Appl.* **2024**, *83*, 56909–56938. [CrossRef]

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.