applied sciences

MDPI

Article

SecureVision: An Open-Source User-Customizable Image Encryption Program

Mehrdad Shahmohammadi Beni [1,2], Hiroshi Watabe [1] and Kwan Ngok Yu [2,*]

[1] Division of Radiation Protection and Safety Control, Cyclotron and Radioisotope Center, Tohoku University, 6-3 Aoba, Aramaki, Aoba-ku, Miyagi, Sendai 980-8578, Japan; ben.sh@my.cityu.edu.hk (M.S.B.); watabe@cyric.tohoku.ac.jp (H.W.)
[2] Department of Physics, City University of Hong Kong, Tat Chee Avenue, Kowloon Tong, Hong Kong, China
* Correspondence: peter.yu@cityu.edu.hk; Tel.: +852-3442-7812

Abstract: Data security has become indispensable, with a view to keep sensitive information confidential. One important method is through image encryption, upon which features in an image would no longer be visible. The original image with its features could only be restored upon decryption using a set of keys. There are prestigious works in the literature regarding image encryption. However, there is a lack of easy-to-use, GUI-based, user-customizable computer programs for image encryption. In the present work, we developed a GUI-based image encryption and decryption program with server file transfer support, namely, SecureVision. A custom-made random number generator using the equation of an ellipse was developed to randomly shuffle the pixel positions. SecureVision was found to be robust, user-friendly and fast in both encryption and decryption. The program was highly sensitive to the supplied keys, which prevented brute-force attacks. SecureVision provided full user control, where users could modify the program modules to match their desired applications, which was particularly desirable for pedagogical purposes in that interested parties had the freedom to explore the concept of image encryption and decryption. SecureVision is distributed under a GPLv3 license, which would allow everyone to use, modify and distribute the program without any restriction.

Keywords: image encryption; data protection; open-source program; software; random numbers

check for updates

Citation: Shahmohammadi Beni, M.; Watabe, H.; Yu, K.N. SecureVision: An Open-Source User-Customizable Image Encryption Program. *Appl. Sci.* **2021**, *11*, 7915. https://doi.org/10.3390/app11177915

Received: 11 July 2021
Accepted: 24 August 2021
Published: 27 August 2021

1. Introduction

Data security and protection have become indispensable, with a view to keep sensitive and personal information confidential. There are myriad ways to protect data, and an important one is through encryption. Broadly speaking, encoding information from plaintext to ciphertext format would be referred to as data encryption. This would be very useful when storing or sending information over the network, as the ciphertext format of the transferred information would be effectively meaningless to an unauthorized person. The encrypted data could then be decrypted to view the plaintext (i.e., original contents) by an authorized user.

Through this concept, one can encrypt images to protect their contents. One main and important application of image encryption is in the field of medical imaging. In fact, the popularity of digital medical imaging [1–3] has rendered patients' confidential images at risk, so it is well-justified to encrypt images with algorithms which are difficult to be deciphered by unauthorized individuals. Generally speaking, the positions and values of pixels will be altered in an encrypted image, in which the features will no longer be recognizable [4–9]. Only with the use of correct keys would it be possible for one to restore the original image [10–14]. Many image encryption algorithms were developed by previous investigators [1,15]. For example, an image encryption algorithm using Arnold transform coupled with random strategies was proposed [16]. On another front, the block shuffling technique was introduced [17], where the image was divided into overlapping blocks which were then shuffled to encrypt the image. In another study [18], an interesting approach using a skew-tent chaotic map was used for image encryption, without the need of changing the pixel information. The pixel bit method based on the chaotic map was previously investigated [19], which used a single chaos map to evenly distribute the pixel values from 0 to 255. The algorithm was found promising in terms of large key space and hiding ability. In another study, Alawida et al. [20] introduced an enhancement to the tent map by hybridizing it with the deterministic finite state machine, which aimed to maximize the diffusion and confusion properties of the encryption technique. The random scrambling of pixel data through position shuffling was also studied [21], and the results were found promising. Recently, Liang et al. [22] used the elliptic curve cryptography (ECC) to develop a public key image encryption method, which was shown to be highly secured against statistical analysis attacks. In another recent study [23], an improved AES algorithm based on the use of synchronized dynamic keys was introduced. Conventional AES cryptosystems used a static key that should be kept private. Upon introducing the synchronization concept, a dynamic and random key would replace the traditional static key, which significantly enhanced the security of encrypted images. Li et al. [24] introduced a novel chaos-based image encryption method through the use of DNA-encoding and plaintext permutations. This work proposed an interesting concept in which the plain image was encoded into a nucleotide sequence. The authors demonstrated an assuring security performance of their proposed method that could enhance the digital image security. Apart from these studies, various other methods were proposed, and interested readers are referred to References [25–27] and references therein for more information. Many conventional encryption techniques such as the AES algorithm are very useful. For example, Reddy and Rao developed a GUI-based image encryption program on the secured TFTP protocol and employed the AES algorithm for encryption [28]. However, many of these techniques are sophisticated, which presents a challenge for students and young researchers in the field. The open-source program described in this manuscript used custom-built modules, so the users would not be restricted to conventional methods and algorithms and could focus on exploring and experimenting with the concept of image encryption, which presents an advantage from the pedagogical perspective.

Considering the importance of data protection and security, it is pertinent to have a versatile and easy-to-use tool that can encrypt and decrypt images in a secured manner. In addition, it would also be beneficial to have a highly customizable program which allows users to modify the program modules. To the best of our knowledge, there are currently no image encryption tools that provide such full control, robustness, stability and ease of use. In the present work, a versatile open-source computer program for image encryption, named SecureVision, was developed. The present computer program gives full control to users in all encryption and decryption steps. This also provides an additional advantage from the pedagogical point of view. One main advantage of the present work is its simplicity when compared to Salsa20 or ChaCha stream ciphers, which could hopefully ease the challenging tasks of teaching and learning image encryption and decryption.

SecureVision randomizes pixel positions and shuffles pixel values to encrypt images. It also has server support to securely transfer encrypted images through a secured shell (SSH) protocol to a designated server. In the present work, we introduced, discussed and benchmarked SecureVision. The executable program, source codes and all numerical examples presented in this paper were made available publicly under a GPLv3 license, which would allow everyone to use, modify and distribute without any restrictions.

2. Materials and Methods

2.1. Random Number Generator

A custom-made random number generator based on the equation of ellipse was developed. This was needed because the random sequences needed to be reproducible upon image decryption, when the user input the correct keys used for image encryption. In this method, the keys were actually the x-coordinate, and major and minor axes of an ellipse, where the decimal part of the y-coordinate would be used (see Figure 1). As shown in Figure 1, for a specific major axis a, minor axis b and x-coordinate, there would be a specific y-coordinate value. The equation to determine the y-coordinate value is:

$$y = \sqrt{b^2 - \frac{b^2}{a^2}x^2}\,. \quad (1)$$

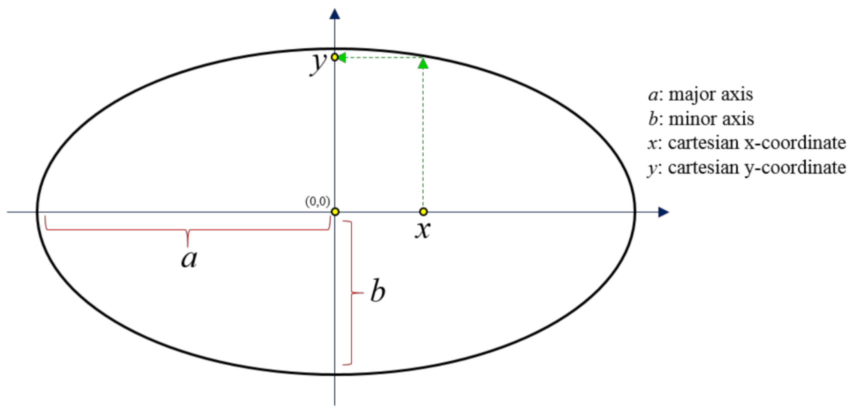

Figure 1. Schematic diagram showing an ellipse with major and minor axes, centered at the origin of the Cartesian coordinate system.

The obtained y-coordinate was then multiplied with the remainder of y and *deno*, which could be represented as mod(y,*deno*). The value of *deno* itself is another user-defined key that requires the user to input.

The program changed the minor axis b of the ellipse according to the width and height of the image loaded into the program for encryption. The decimal part of the y-coordinate value multiplied by the modulo operation as discussed above was used as a random number between 0 and 1. It is remarked that the absolute value was taken. To assess the randomness of our generated results, we compared these with the built-in uniform random number generator in FORTRAN90 programming language (see Figure 2). The results shown in Figure 2 were for 1000 iterations. The random numbers generated from our model were compared against those generated by FORTRAN90, and the randomness of both sets of random numbers were similar. Moreover, we evaluated the distribution of differences between the random numbers generated by our model and those from the standard FORTRAN90 routine, which is shown in Figure 3.

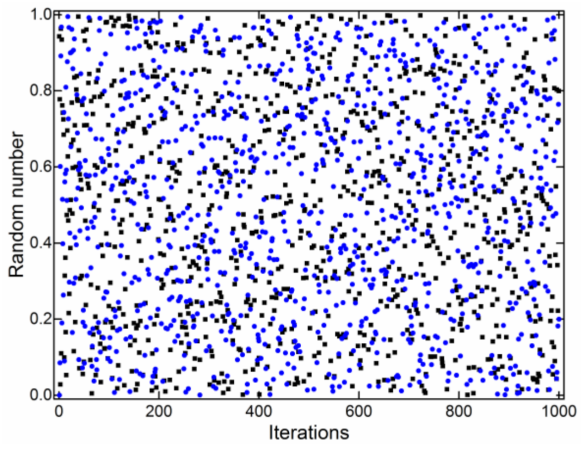

Figure 2. Graphical comparison between random numbers generated using our model (black squares) and the FORTRAN90 compiler (blue circles) for 1000 iterations.

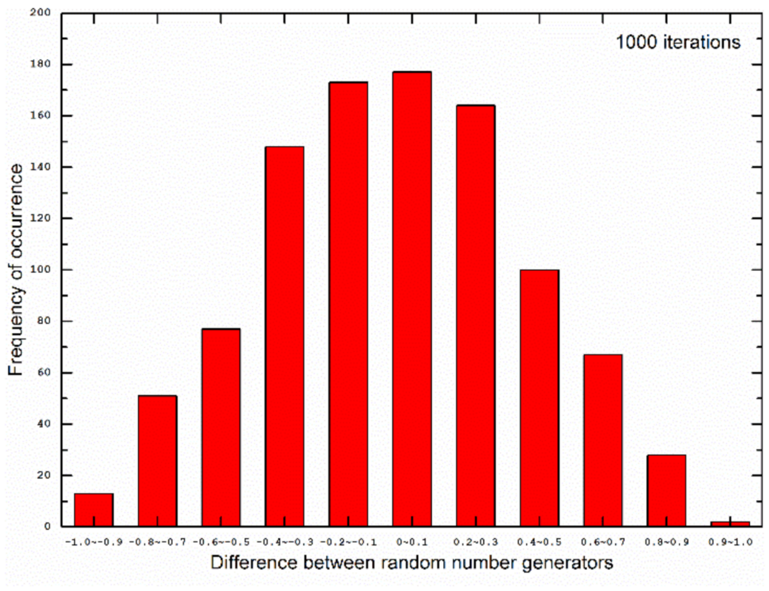

Figure 3. Distribution of differences between random numbers generated from our model and the standard FORTRAN90 routine for 1000 iterations, with a bin size of 0.1 for the differences.

Figures 2 and 3 show that the pseudo-random numbers generated from our model and the standard FORTRAN90 routine were similar. The peak of the distribution of differences lay in the bin with a range from 0 to 0.1. The mean of the differences was found to be -7.55×10^{-3}, which was very close to zero.

In addition to the comparison shown in Figures 2 and 3, a more extensive benchmarking using a numerical example was performed. Here, we computed the value of π by way of Monte Carlo simulations using the two random number generators. The source code for these Monte Carlo simulations can be found at: https://figshare.com/articles/software/SecureVision_A_versatile_open-source_image_encryption_program_for_students/14946054 (accessed on 21 August 2021). The variables were double-precision 8-byte real numbers (i.e., real*8 type), and the code was written in standard FORTRAN90 and compiled using Gfortran (GNU Fortran). From 10^6 iterations, the values of π were determined to be 3.1411 and 3.1416 respectively, using random number generators from our model and the standard FORTRAN90 routine, which could be considered close. A computer routine in the C++ language, which is one of the commonly used programming languages, has been used to provide data for comparing the randomness. From 106 iterations, the values of π were determined to be 3.1424 and 3.1414 respectively, using random number generators from our model and the standard C++ routine. Although the random number generator from our model was not strictly comparable to that from the standard FORTRAN90 routine in terms of applications, it was not a critical issue as our goal was to reproduce the randomness upon entering the correct set of keys. However, using the random number generator from our model for other applications which require absolute randomness is not recommended. The frequency (monobit) randomness test of the random number generator was used in the present work. Bits 0 and 1 are obtained for random numbers smaller and greater than 0.5, respectively. In this test, 10^4 blocks, each with a size of 10^4, were used. The p-value was calculated as in [29]:

$$p\text{-value} = \operatorname{erfc}\left(\frac{|\text{diff}|}{\sqrt{n}} \div \sqrt{2}\right) \quad (2)$$

where diff is the difference between the counted 0 and 1 bits, and n is the total number of counts for each block. The obtained p-values and their interpretation with a significance level of 0.01 are shown in Table 1. A p-value of 1 corresponds to a perfectly random tested sequence, while a p-value of 0 corresponds to non-randomness [30].

Table 1. Results from the frequency (monobit) randomness test.

***p*-Value**	**0.0001–0.001**	**0.001–0.01**	**0.01–0.1**	**0.1–1.0**
Count	7	83	904	8932
Results	Non-random	Non-random	Random	Random

2.2. Random Shuffling of Pixel Positions

The positions of pixels in an image were randomly shuffled using the random number generator from our model. The width (w) and height (h) of the image were determined from the image file, which was in a greyscale pgm format at the current stage. The random pixel shuffling scheme is shown schematically in Figure 4.

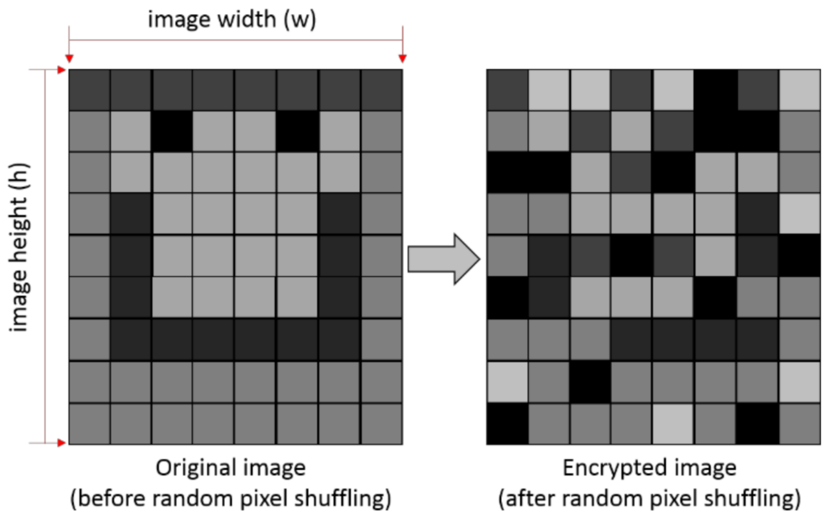
image width (w)
image height (h)
Original image
(before random pixel shuffling)
Encrypted image
(after random pixel shuffling)

Figure 4. Schematic diagram showing random shuffling of pixel positions using the random number generator from our model.

The random numbers generated from our model and the standard FORTRAN90 routine were floating-point numbers between 0 and 1. Our main goal here was to shuffle the pixels along both long and short edges of the image, so two columnar arrays for random width and height positions should be generated by way of floating-point random numbers (between 0 and 1) generated by the model. A 2-dimensional array named "INP" was created with the pixel components of the original image, INP(i,j), as input elements, where i and j are width and height positions of the original image. The generated random numbers were used to shuffle the elements in INP to create a new array. It is remarked here that conversion of floating-point random numbers to integers for the width and height positions of the image could cause overlaps in the shuffled pixel positions. This issue will be discussed and tackled in later parts of the present paper. Shuffling of pixel positions would in effect hide the features in an image and therefore encrypt it. An example is shown in Figure 5, where the original image has been encrypted only by way of shuffling the pixel positions. The corresponding histograms showing the distribution of greyscale values of the pixels are also shown, in which the x-axis represents greyscale values ranging from 0 (black) to 255 (white). The features in the original image could no longer be identified upon random shuffling of pixel positions, and could only be restored upon decryption using the set of keys employed for the encryption. The random number generator from our model handled the decryption and generated the same set of random numbers for the same set of keys. The possibility of a brute-force attack to decipher the private keys by an unauthorized user would be reduced by employing longer keys during encryption, which would in turn enhance the security of the encrypted images.

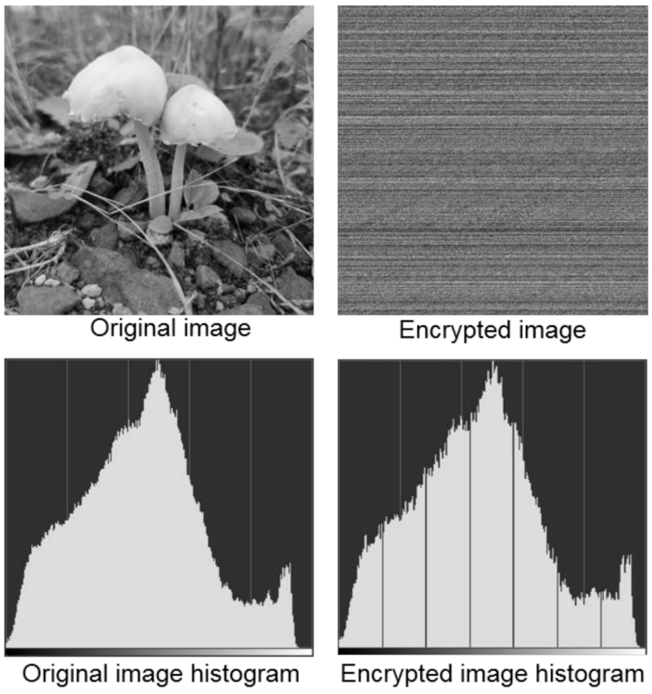

Figure 5. Original and encrypted images with their respective histograms showing the distribution of greyscale values of pixels. The x-axis in the histograms represents greyscale values ranging from 0 (black) to 255 (white).

A descriptive algorithm that describes part of the encryption method employed in the present work is shown below, Algorithm 1.

Algorithm 1 Random Pixel Shuffling

```
1. Initialize the variables.
2. Read image size, width ($w$) and height ($h$).
3. Read the private keys: $x, a, b$ (see Equation (1)), $deno, izmax, delta,$ A (see Equation (3)).
4. Allocate 2 × 2 array, INP.
5. Read image components into INP.
6. Calculate the $y$-coordinate of ellipse based on private keys
7. Calculate modulo of the $y$-coordinate, $deno$.
8. Multiply the $y$ value with the remainder of the division
9. Calculate the positive decimal value from step 8, obtain random number ($z$).
10. allocate two 2 × 2 arrays containing horizontal (pw) and vertical (ph) image coordinates
11. convert $z$ into image coordinates and read into pw and ph.
12. Allocate 2 × 2 array, INPC.
13. shuffle the image pixels based on pw and ph.
14. store the shuffled pixel values in INPC array.
15. write out INPC into an image file.
16. $deno + delta$, then $n = n + 1$.
17. if ($n < izmax$), go to step 6, if ($n > izmax$), then terminate, maximum iterations ($izmax$) reached.
```

2.3. *Shuffling of Pixel Values*

The pixel values would remain unchanged if only the pixel positions of an image were shuffled (see Figure 5). In other words, the histograms of the original and encrypted image would look almost identical, as shown in Figure 5. This was undesirable for image encryption, as an unauthorized user could reconstruct the original image using the data from the histogram. To avoid this, the histogram of the encrypted image should either be very different from that of the original image or in ideal cases, be uniform and flat.

In the present version of our program, we shuffled the pixel values through a cosine function that generated pixel values which were then summed with the pixel values in the encrypted image. This was mathematically represented as:

$$\mathbf{P}(i,j) = 255 \times |A \times \cos(k)|\ k = 1, 2, \ldots, w \times h\,, \tag{3}$$

where $\mathbf{P}(i,j)$ was an array with the same size as the encrypted image pixel array and A was a user-defined variable that controlled the amplitude of the cosine function. The absolute value of the product of the cosine function with A was then multiplied by 255 (i.e., the largest pixel value in a greyscale image). In fact, as the present computer code is open-source, users can experiment with different functions and variables to obtain their desired level of encryption. We examined histograms of encrypted images for three different A values, as shown in Figure 6. It could be noticed that the histograms were no longer similar to the histogram of the original image shown in Figure 5. Therefore, the proposed pixel value shuffling method worked satisfactorily.

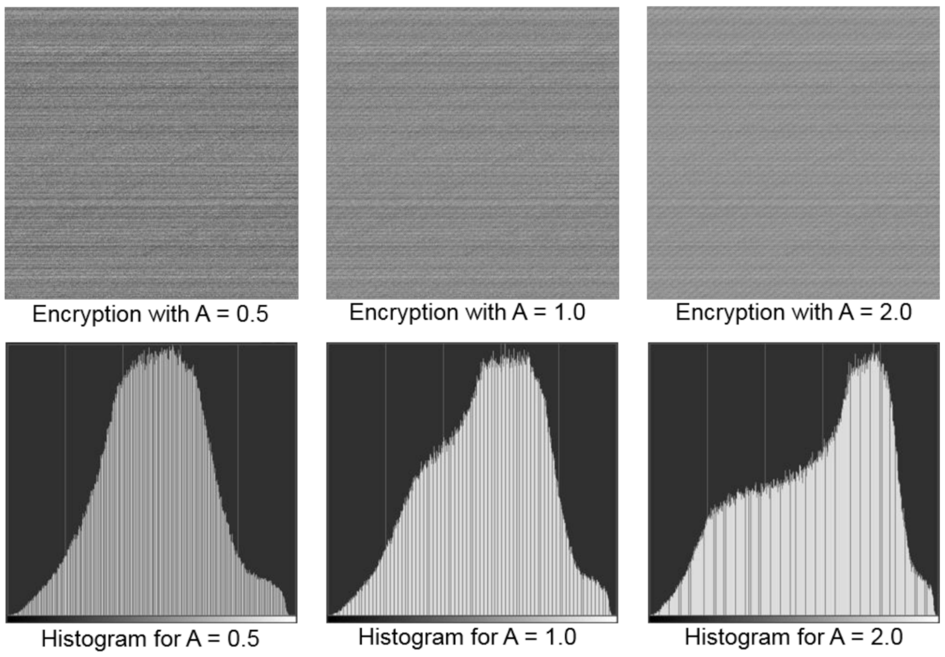

Figure 6. Original and encrypted images with their respective histograms showing the distribution of greyscale values of pixels. The *x*-axis in the histograms represents greyscale values ranging from 0 (black) to 255 (white).

3. Results and Discussion

3.1. Decryption and Merging

Decryption of the encrypted images should be relatively simple, considering that the same set of keys used for encryption should be presented to the program. In fact, the decryption step essentially returned the shuffled pixels back to their original positions and restored their original values. However, as mentioned above, due to changes between floating-point and integer variables, as well as random position shuffling, there could be pixel overlaps during encryption. This pixel overlap led to a loss of data in some parts of the encrypted image, which subsequently caused a loss of pixels in the restored image. For example, decryption of the image shown in Figure 6 with A = 1.0 is shown in Figure 7, where loss of pixels can be clearly seen.

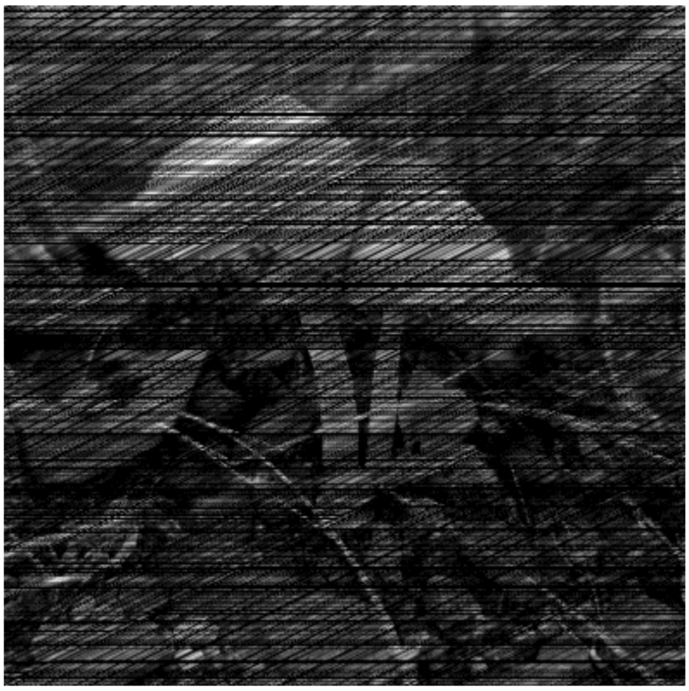

Figure 7. Restored image after decryption of the image shown in Figure 6 with A = 1.0. The black lines in the image represent pixel loss regions.

To tackle this problem, we developed a module in our program that performed multi-layer merging of decrypted images with different pixel offsets. In this module, multiple encrypted images were produced with different *deno* and b values (see Section 2), which led to small offsets for the shuffled pixel positions. The offset value was defined by the user, which generated a small range of variations in the random numbers so that those missing pixel positions could be reconstructed when different images with different offsets were merged together. The offset value was one of the keys for image encryption and decryption. The evolution of decryption and multi-layer merging is shown in Figure 8, which demonstrates that our program can successfully decrypt and recover the original image (see Figure 5 for the original image).

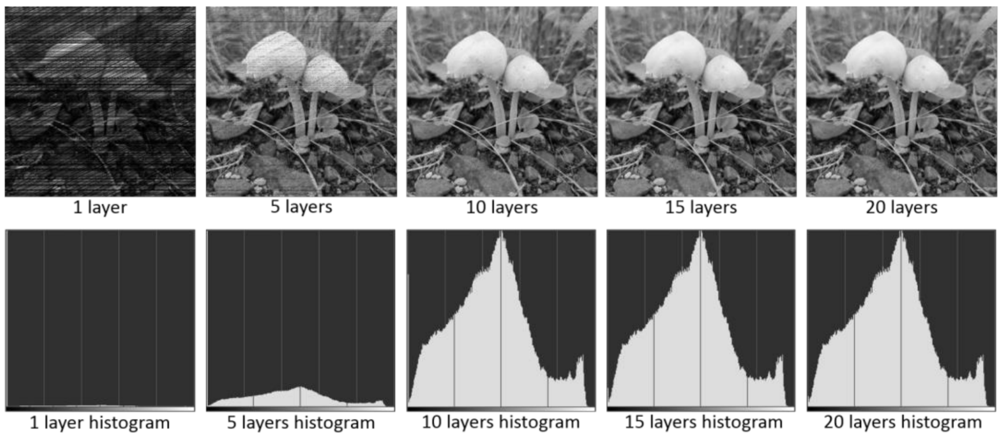

Figure 8. Evolution of image decryption with multi-layer merging. The associated histogram showing the distribution of greyscale values of pixels for each decrypted image is also shown. The x-axis in the histograms represents greyscale values ranging from 0 (black) to 255 (white).

The requirement of the set of keys as well as the multiple encrypted image layers to fully recover the original image in fact added an extra layer of security to the current method. The multiple encrypted images could effectively be treated as public keys, while the values used by the user to encrypt the image could be treated as private keys.

3.2. Graphical User Interface

A graphical user interface (GUI) was developed for the present computer program, which facilitated users to encrypt, decrypt and transfer their images in a very efficient and easy-to-use manner. The GUI implementation of any software leads to a user-friendly working interface. The GUI can be broken down into three main sections, namely: (1) encryption, (2) decryption and (3) transfer to server. A screenshot of the developed GUI is shown in Figure 9.

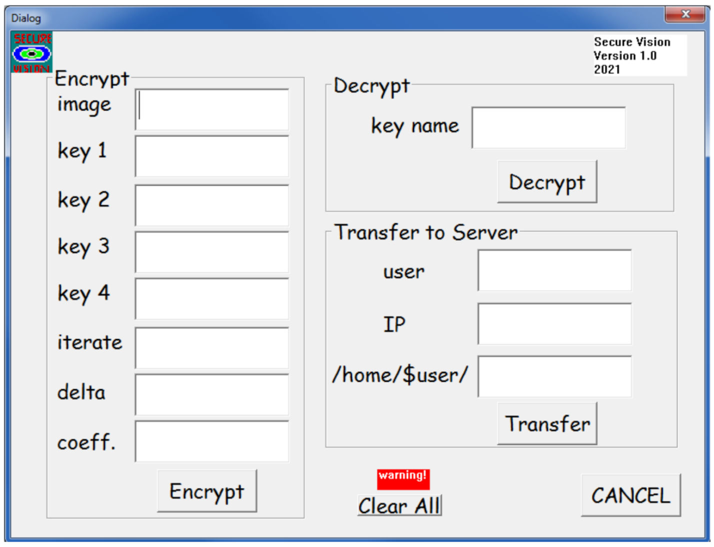

Figure 9. The developed graphical user interface (GUI) for encryption and decryption images, and for data transfer to server.

The encryption part of the program took eight (8) user inputs, namely, (1) filename of the image that needed to be encrypted, (2) key 1, which was the x value for the random number generator, (3) key 2, which was value of a for the random number generator, (4) key 3, which was value of b for the random number generator, (5) key 4, which was the value of *deno* for the random number generator, (6) iteration number for multi-layer merging, (7) delta value for pixel offset and (8) coefficient, which was the value of A in Equation (2) for histogram shuffling. The GUI read the user inputs and executed the encoder program to encrypt the images with the user input keys.

The decryption part of the program read the file which stored the user parameters for encryption, with the filename (key name) specified by the user. In the current version of the program, the key file is an ASCII data file with the ".dat" extension. This key file could be further encrypted with standard encryption techniques and software such as GNU Privacy Guard (commonly known as GPG) [31].

The last section of the GUI program concerned transfers of files, which employed the secured file transfer protocol (through SSH connection) to copy the encrypted images to a server or computer. The user was required to input the username, server IP address and a directory in their home user folder. The Microsoft Windows version of our program employed the PSCP protocol to transfer the encrypted images to the server. This automatic function would be very useful for quick and secured file transfer to (1) backup servers for further data protection or (2) servers for sharing data in a secured manner with authorized users.

3.3. Examples and Testing

To extensively check the present program, we performed a number of tests using images with different sizes and resolutions. Five (5) example images were tested, each with different features and scenery. The original, encrypted and decrypted images are shown in Figures 10–14. In addition, the original, encrypted and decrypted images with the key values that were used in the present program can be downloaded from: https://figshare.com/articles/software/SecureVision_A_versatile_open-source_image_encryption_program_for_students/14946054 (accessed on 21 August 2021).

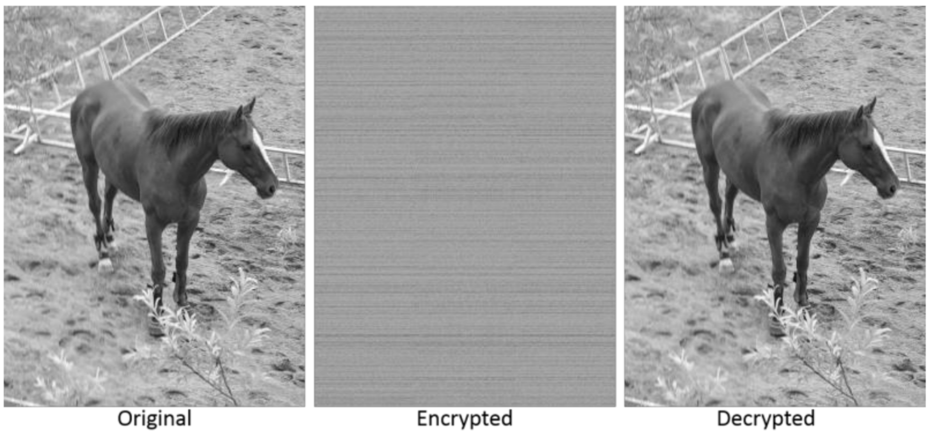

Figure 10. A 960 × 1280 photograph example used for testing the present program. Original, encrypted and decrypted images are shown.

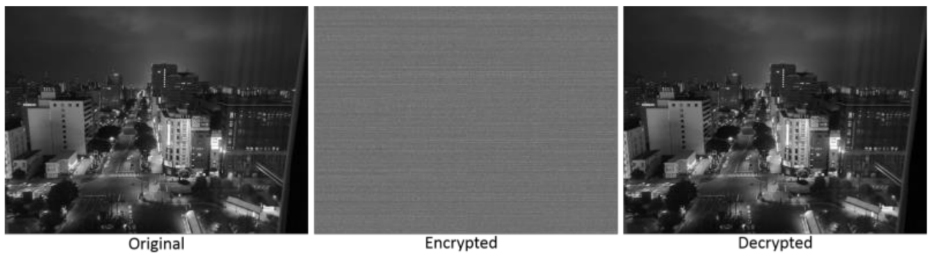

Figure 11. A 1280 × 960 photograph example used for testing the present program. Original, encrypted and decrypted images are shown.

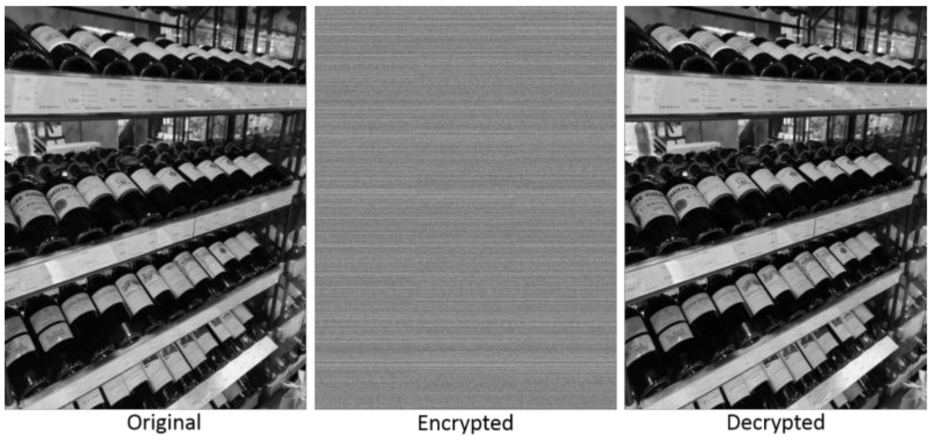

Figure 12. A 960 × 1280 photograph example used for testing the present program. Original, encrypted and decrypted images are shown.

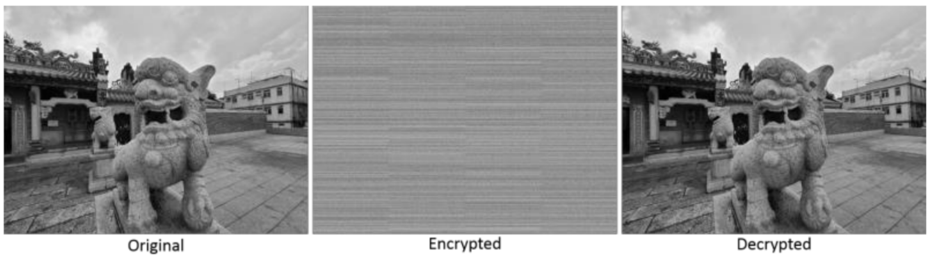

Figure 13. A 1280 × 960 photograph example used for testing the present program. Original, encrypted and decrypted images are shown.

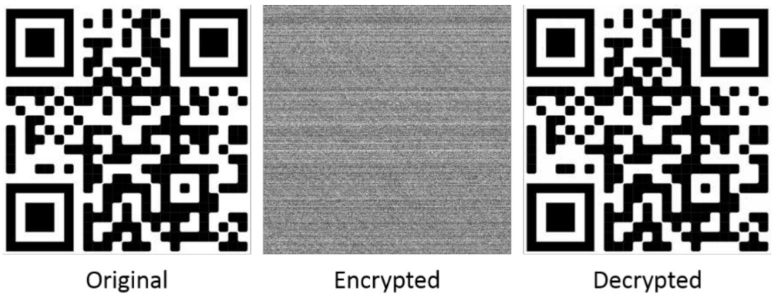

Figure 14. A 714 × 715 QR code image example used for testing the present program. Original, encrypted and decrypted images are shown (The QR code was generated on 3 July 2021 that directed to the main website of the City University of Hong Kong).

Figures 10–14 show that the present program can satisfactorily encrypt and decrypt images. There were no significant losses of pixels or resolution in the decrypted images. The QR code example shown in Figure 14 demonstrates that the present program can fulfill stringent decryption requirements, since the restored QR code would not have worked in case of incorrect pixel positions or significant pixel losses.

A comparative analysis between image encryption and decryption using our model and those using previously developed methods has been performed. The results from the previous works of Ye [19] and Hua et al. [1] were used in this comparison. The method proposed by Ye [19] was based on pixel scrambling based on the chaotic map, while Hua et al. [1] employed high-speed scrambling and pixel adaptive diffusion to encrypt images. Comparisons between their results and the results from our model are shown in Figures 15 and 16.

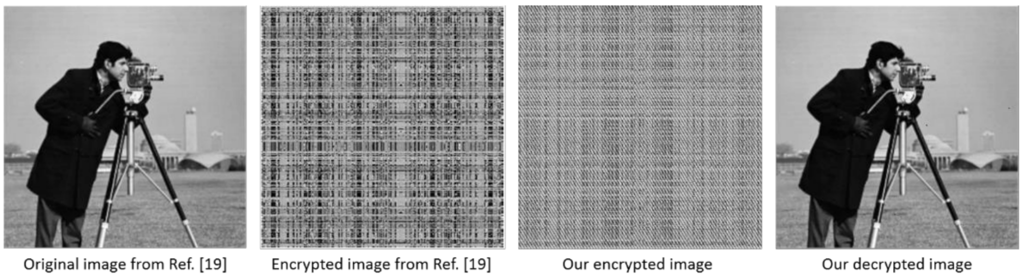

Figure 15. Comparison between the encrypted images from previous work of Ye [19] and our method. The decrypted image using our method is also shown.

The encryption and decryption times for a number of images with different resolutions are shown in Table 2. It is remarked that the reported duration is the overall time for 10 image layers (i.e., the program produces 10 encrypted images and merges these 10 image layers during decryption).

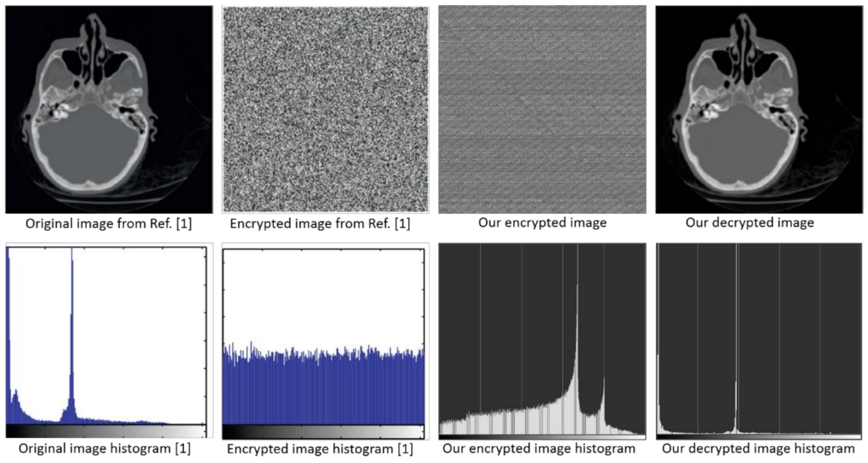
Original image from Ref. [1]
Encrypted image from Ref. [1]
Our encrypted image
Our decrypted image
Original image histogram [1]
Encrypted image histogram [1]
Our encrypted image histogram
Our decrypted image histogram

Figure 16. Comparison between the encrypted images from previous work of Hua et al. [1] and our method. The histogram for every image is shown below each figure.

Table 2. Encryption and decryption times for different image resolutions (time shown is the total time taken for 10 image layers).

Image Size	Encryption Time (min:s:ms)	Decryption Time (min:s:ms)
768 × 1024	00:05:53	00:07:39
1200 × 1600	00:13:42	00:17:95
2367 × 1957	00:32:56	00:43:06
2760 × 2184	00:42:67	00:56:92
3024 × 4032	01:29:20	02:00:32

Comparisons of some features (key space, GUI, speed and programming language) of techniques used in selected previous investigations and in the present work are shown in Table 3.

Table 3. Comparisons of some features (key space, GUI, speed and programming language) of techniques used in selected previous investigations and in the present work.

Ref. No.	Technique	Key Space	GUI	Speed	Programming Language
[32]	Bitplane decomposition and Chaotic maps	0.25×10^{64}	No	Low	Matlab
[30]	Permutation and interrelated chaos	10^{108}	No	Low	-
[33]	Dynamic random growth technique	$>10^{96}$	No	Moderate	C++
[34]	Swapping-based confusion approach	0.18×10^{60}	No	Good	C
[35]	Arnold map	$>2^{100}$	No	Moderate	Matlab
[36]	Logistic mapping	10^{112}	No	Moderate	Matab
[19]	Scrambling encryption algorithm based on chaos map	$>2^{100}$	No	Moderate	Matab
[28]	AES on secured TFTP protocol	$>2^{128}$	Yes	Moderate	Python
Our work	Random pixel shuffling	$>10^{100}$	Yes	Good	Fortran

3.4. Decryption Sensitivity to Keys

To ascertain the security provided by the present program, we assessed the resilience of the method against brute-force attack by assessing the sensitivity of the program to the keys supplied to the decryption module. For this assessment, we assumed all the key values were obtained correctly through a brute-force attack, except one single key value which slightly deviated from the correct value. The images in Figures 10 and 14 were used as examples and the decrypted images are shown in Figures 17 and 18, respectively. It was shown that even a slight deviation in a single key value would result in a total loss of features in the decrypted image, which demonstrated the resilience of the method against brute-force attack and ensured security of confidential images.

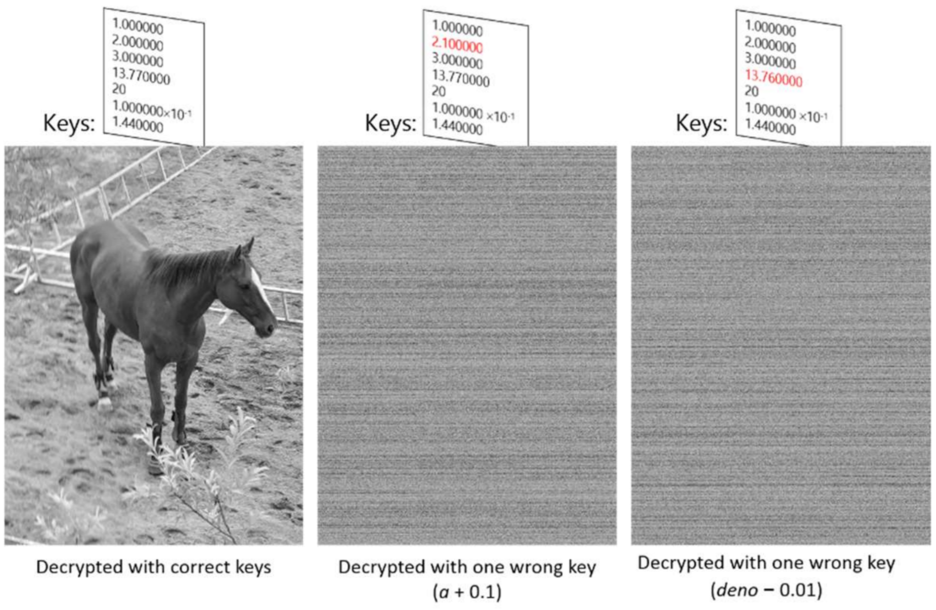
Keys: 1.000000 2.000000 3.000000 13.770000 20 1.000000×10⁻¹ 1.440000
Keys: 1.000000 2.100000 3.000000 13.770000 20 1.000000 ×10⁻¹ 1.440000
Keys: 1.000000 2.000000 3.000000 13.760000 20 1.000000 ×10⁻¹ 1.440000
Decrypted with correct keys
Decrypted with one wrong key (a + 0.1)
Decrypted with one wrong key (deno − 0.01)

Figure 17. Decryption of encrypted image in Figure 10 using correct and wrong keys. The key values used for each decrypted image are shown above the image, with the wrong key value shown in red.

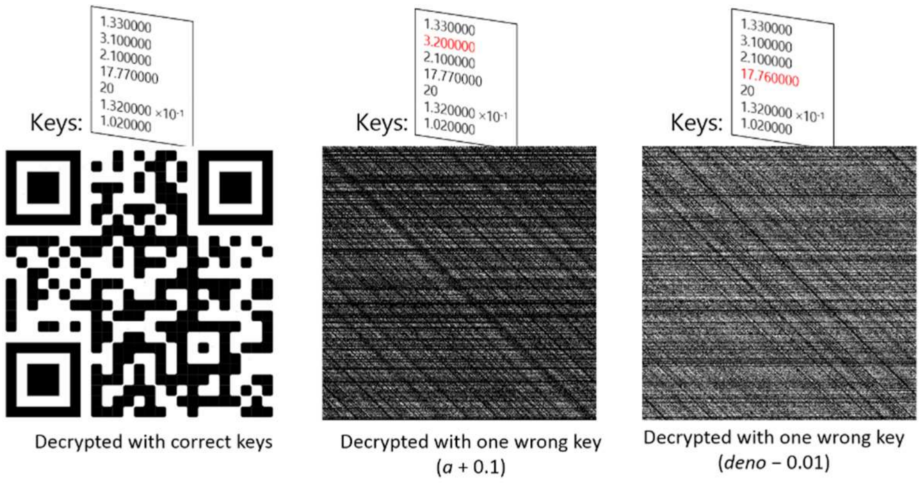
Keys: 1.330000 3.100000 2.100000 17.770000 20 1.320000 ×10⁻¹ 1.020000
Keys: 1.330000 3.200000 2.100000 17.770000 20 1.320000 ×10⁻¹ 1.020000
Keys: 1.330000 3.100000 2.100000 17.760000 20 1.320000 ×10⁻¹ 1.020000
Decrypted with correct keys
Decrypted with one wrong key (a + 0.1)
Decrypted with one wrong key (deno − 0.01)

Figure 18. Decryption of encrypted image in Figure 14 using correct and wrong keys. The key values used for each decrypted image are shown above the image, with the wrong key value shown in red.

4. Conclusions

In present work, we developed an open-source image encryption program named "SecureVision". The program can encrypt and decrypt black and white images. The developed GUI allows users to easily interact with the program and transfer data through a secured shell connection to a designated server. Random shuffling of pixel positions was controlled by a random number generator with randomness controlled by user-defined keys. Upon supplying the same keys to the decryption module of the program, the shuffled pixels were restored to their original positions. In addition, the pixel values were shuffled during encryption to further enhance the security. The present program was found to be sensitive to the keys supplied by the user, which prevented brute-force attack by an unauthorized user. The open-source nature of the program would allow users to modify the code for their desired applications. In addition, the program would be a useful tool for students and those parties who are interested in learning image encryption and decryption. The obtained results were in good agreement with those obtained in previously published studies. The randomness of pixel shuffling was statistically verified. The time taken for encryption and decryption confirmed the good speed of the present method. The present program and the introduced method were also valuable from the pedagogical point of view. We aim to extend the present program to encrypt and decrypt color images through similar shuffling of pixel positions and values (RBG components) in future works. Furthermore, more server support features will be implemented so that users would have more control in their data transfer and backup. In addition, we would like to investigate the possibility of parallel implementation of the present method using MPI (Message Passing Interface) or CUDA (Compute Unified Device Architecture).

Author Contributions: Conceptualization, M.S.B.; methodology, M.S.B. and K.N.Y.; software, M.S.B.; validation, M.S.B. and K.N.Y.; formal analysis, M.S.B.; investigation, M.S.B.; resources, M.S.B., H.W. and K.N.Y.; data curation, M.S.B.; writing—original draft preparation, M.S.B.; writing—review and editing, H.W. and K.N.Y.; visualization, M.S.B.; supervision, K.N.Y.; project administration, H.W. and K.N.Y.; funding acquisition, H.W. and K.N.Y. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the Special Grant for the Development of Virtual Teaching and Learning (VTL), No. 6430120, from the University Grants Committee of Hong Kong SAR, and by Grants-in-Aid for Scientific Research, No. 20H03615, from the Ministry of Education, Culture, Sports, Science and Technology (MEXT), Japanese Government.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Data supporting reported results can be found at https://figshare.com/articles/software/SecureVision_A_versatile_open-source_image_encryption_program_for_students/14946054 (accessed on 21 August 2021).

Conflicts of Interest: The authors declare no conflict of interest. The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript, or in the decision to publish the results.

Abbreviations

AES	Advanced Encryption Standard
ASCII	American Standard Code for Information Interchange
CUDA	Compute Unified Device Architecture
GPG	GNU Privacy Guard
GUI	Graphical User Interface
IP	Internet Protocol
MPI	Message Passing Interface
PSCP	PuTTY Secure Copy Client
QR code	Quick Response code
RGB	Red Green Blue
SSH	Secure Shell
a	Major axis of ellipse
b	Minor axis of ellipse

x	Cartesian x-coordinate
y	Cartesian y-coordinate
deno	Denominator for modulo operation
w	Image width
h	Image height
INP(i,j)	Original image element array
A	Cosine function amplitude controller

References

1. Hua, Z.; Yi, S.; Zhou, Y. Medical image encryption using high-speed scrambling and pixel adaptive diffusion. *Signal Process.* **2018**, *144*, 134–144. [CrossRef]
2. Zinger, S.; Ruijters, D.; Do, L.; de With, P.H.N. View interpolation for medical images on autostereoscopic displays. *IEEE Trans. Circuits Syst. Video Technol.* **2011**, *22*, 128–137. [CrossRef]
3. Lacoste, C.; Lim, J.H.; Chevallet, J.P.; Le, D.T. Medical-image retrieval based on knowledge-assisted text and image indexing. *IEEE Trans. Circuits Syst. Video Technol.* **2007**, *17*, 889–900. [CrossRef]
4. Chai, X.; Chen, Y.; Broyde, L. A novel chaos-based image encryption algorithm using DNA sequence operations. *Opt. Lasers Eng.* **2017**, *88*, 197–213. [CrossRef]
5. Zhang, Y.Q.; Wang, X.Y. A new image encryption algorithm based on non-adjacent coupled map lattices. *Appl. Soft Comput.* **2015**, *26*, 10–20. [CrossRef]
6. Ping, P.; Xu, F.; Wang, Z.J. Image encryption based on non-affine and balanced cellular automata. *Signal Process.* **2014**, *105*, 419–429. [CrossRef]
7. Chai, X.; Gan, Z.; Chen, Y.; Zhang, Y. A visually secure image encryption scheme based on compressive sensing. *Signal Process.* **2017**, *134*, 35–51. [CrossRef]
8. Li, X.W.; Lee, I.K. Modified computational integral imaging-based double image encryption using fractional Fourier transform. *Opt. Lasers Eng.* **2015**, *66*, 112–121. [CrossRef]
9. Liu, H.; Wang, X. Image encryption using DNA complementary rule and chaotic maps. *Appl. Soft Comput.* **2012**, *12*, 1457–1466. [CrossRef]
10. Gong, L.; Liu, X.; Zheng, F.; Zhou, N. Flexible multiple-image encryption algorithm based on log-polar transform and double random phase encoding technique. *J. Mod. Opt.* **2013**, *60*, 1074–1082. [CrossRef]
11. Wang, X.Y.; Zhang, Y.Q.; Bao, X.M. A novel chaotic image encryption scheme using DNA sequence operations. *Opt. Lasers Eng.* **2015**, *73*, 53–61. [CrossRef]
12. Zhou, N.; Hu, Y.; Gong, L.; Li, G. Quantum image encryption scheme with iterative generalized Arnold transforms and quantum image cycle shift operations. *Quantum Inf. Process.* **2017**, *16*, 164. [CrossRef]
13. Zhang, Y.Q.; Wang, X.Y. A symmetric image encryption algorithm based on mixed linear–nonlinear coupled map lattice. *Inf. Sci.* **2014**, *273*, 329–351. [CrossRef]
14. Hua, Z.; Zhou, Y.; Pun, C.M.; Chen, C.P. 2D Sine Logistic modulation map for image encryption. *Inf. Sci.* **2015**, *297*, 80–94. [CrossRef]
15. Tang, Z.; Yang, Y.; Xu, S.; Yu, C.; Zhang, X. Image encryption with double spiral scans and chaotic maps. *Secur. Commun. Netw.* **2019**, *2019*, 8694678. [CrossRef]
16. Tang, Z.; Zhang, X. Secure image encryption without size limitation using Arnold transform and random strategies. *J. Multimed.* **2011**, *6*, 202. [CrossRef]
17. Tang, Z.; Zhang, X.; Lan, W. Efficient image encryption with block shuffling and chaotic map. *Multimed. Tools Appl.* **2015**, *74*, 5429–5448. [CrossRef]
18. Zhang, G.; Liu, Q. A novel image encryption method based on total shuffling scheme. *Opt. Commun.* **2011**, *284*, 2775–2780. [CrossRef]
19. Ye, G. Image scrambling encryption algorithm of pixel bit based on chaos map. *Pattern Recognit. Lett.* **2010**, *31*, 347–354. [CrossRef]
20. Alawida, M.; Teh, J.S.; Samsudin, A. An image encryption scheme based on hybridizing digital chaos and finite state machine. *Signal Process.* **2019**, *164*, 249–266. [CrossRef]
21. Prasad, M.; Sudha, K.L. Chaos image encryption using pixel shuffling. *CCSEA* **2011**, *1*, 169–179. [CrossRef]
22. Liang, H.; Zhang, G.; Hou, W.; Huang, P.; Liu, B.; Li, S. A Novel Asymmetric Hyperchaotic Image Encryption Scheme Based on Elliptic Curve Cryptography. *Appl. Sci.* **2021**, *11*, 5691. [CrossRef]
23. Lin, C.H.; Hu, G.H.; Chan, C.Y.; Yan, J.J. Chaos-Based Synchronized Dynamic Keys and Their Application to Image Encryption with an Improved AES Algorithm. *Appl. Sci.* **2021**, *11*, 1329. [CrossRef]
24. Li, Z.; Peng, C.; Tan, W.; Li, L. A novel chaos-based image encryption scheme by using randomly DNA encode and plaintext related permutation. *Appl. Sci.* **2020**, *10*, 7469. [CrossRef]
25. Cao, X.; Huang, Y.; Wu, H.T.; Cheung, Y.M. Content and privacy protection in JPEG images by reversible visual transformation. *Appl. Sci.* **2020**, *10*, 6776. [CrossRef]
26. Wu, H.; Wang, J.; Zhang, Z.; Chen, X.; Zhu, Z. A Multi-Image Encryption with Super-Lager-Capacity Based on Spherical Diffraction and Filtering Diffusion. *Appl. Sci.* **2020**, *10*, 5691. [CrossRef]

27. Saraiva, D.A.; Leithardt, V.R.; de Paula, D.; Sales Mendes, A.; González, G.V.; Crocker, P. Prisec: Comparison of symmetric key algorithms for IoT devices. *Sensors* **2019**, *19*, 4312. [CrossRef]
28. Reddy, K.R.; Rao, C.M. GUI implementation of image encryption and decryption using Open CV-Python script on secured TFTP protocol. In Proceedings of the AIP Conference, Secunderabad, India, 22–23 December 2017; AIP Publishing LLC: College Park, MD, USA, 2018; Volume 1952, p. 020074. [CrossRef]
29. Zaman, J.K.; Ghosh, R. A review study of NIST statistical test suite: Development of an indigenous computer package. *arXiv* **2012**, arXiv:1208.5740.
30. Wang, X.Y.; Zhang, H.L. A color image encryption with heterogeneous bit-permutation and correlated chaos. *Opt. Commun.* **2015**, *342*, 51–60. [CrossRef]
31. The GNU Privacy Guard. Available online: https://gnupg.org/ (accessed on 5 July 2021).
32. Xu, L.; Li, Z.; Li, J.; Hua, W. A novel bit-level image encryption algorithm based on chaotic maps. *Opt. Lasers Eng.* **2016**, *78*, 17–25. [CrossRef]
33. Wang, X.Y.; Liu, L.T.; Zhang, Y.Q. A novel chaotic block image encryption algorithm based on dynamic random growth technique. *Opt. Lasers Eng.* **2015**, *66*, 10–18. [CrossRef]
34. Chen, J.X.; Zhu, Z.L.; Fu, C.; Yu, H. A fast image encryption scheme with a novel pixel swapping-based confusion approach. *Nonlinear Dyn.* **2014**, *77*, 1191–1207. [CrossRef]
35. Ye, G.D.; Wong, K.W. An efficient chaotic image encryption algorithm based on a generalized Arnold map. *Nonlinear Dyn.* **2012**, *69*, 2079–2087. [CrossRef]
36. Sethi, N.; Sharma, D. A novel method of image encryption using logistic mapping. *Int. J. Comput. Sci. Eng.* **2012**, *1*, 115–119.