Sinkron : Jurnal dan Penelitian Teknik Informatika
Volume 8, Number 1, January 2024
DOI : https://doi.org/10.33395/sinkron.v9i1.13049
e-ISSN : 2541-2019
p-ISSN : 2541-044X

Imperceptible and Robust Encryption: Salsa20 Stream Cipher for Colour Image Data

Aldi Azmi Arfian[1)*], **Christy Atika Sari**[2)], **Eko Hari Rachmawanto**[3)], **Folasade Olubusola Isinkaye**[4)]
[1,2,3)]University of Dian Nuswantoro, Indonesia
[4)]Ekiti State University, Ado Ekiti, Nigeria
[1)]111202214439@mhs.dinus.ac.id, [2)]christy.atika.sari@dsn.dinus.ac.id, [3)]eko.hari@dsn.dinus.ac.id, [4)]folasade.isinkaye@eksu.edu.ng

Submitted: Sep 26, 2023 | **Accepted** : Nov 7, 2023 | **Published** : Jan 1, 2024

Abstract: Data security has become crucial, especially in today's era, therefore we need to protect our personal data to avoid unwanted incidents. The primary objective of this research is to empirically demonstrate the viability of our proposed methodology for encrypting color images using the Salsa20 algorithm, renowned for its stream cipher characteristics, which inherently afford it a swift processing speed. The encryption method we use is to take each pixel from the original image and convert it into bytes based on the RGB value in it, then encrypt it using a keyword that has been converted using a hash function. In this study, we carried out several evaluations to evaluate the performance of the encrypted and decrypted images to test the method we propose, including histogram analysis and compare patterns, visual image testing, and key space analysis. Through this experiment, it has been proven that Salsa20 is effective in maintaining confidentiality and image integrity. Histogram analysis reveals differences in pixel distribution patterns between the original and encrypted images. Visual testing shows that the encrypted image maintains good optical quality. Keyspace analysis ensures the security of encryption keys. The performance evaluation resulted in an NPCR above 99%, UACI had been reached 69.28%, MSE was closes to 0, and the highest PSNR was around 61.89dB, this shows that encrypted images recovered with high accuracy.

Keywords: Data Security; Stream Cipher; Salsa20; Encryption; Hash Function;

INTRODUCTION

In the pursuit of safeguarding data confidentiality, integrity, and availability, the adoption of cryptographic techniques has grown in significance. Cryptography plays a pivotal role in ensuring the secure transmission and storage of data, be it over networks or physical mediums. However, it's crucial to recognize that the landscape of information security is ever-evolving. As technological advancements progress, so do threats and attacks. Thus, continuous research and development in the field of cryptography are essential to create stronger, more efficient, and robust techniques and algorithms. Through the application of robust encryption methods and the careful safeguarding of cryptographic keys, data can be shielded from threats and unauthorized access. It's essential to acknowledge that no security system can provide an absolute guarantee, which fuels the demand for enhanced information confidentiality and motivates the ongoing development of new encryption techniques and algorithms (Jawad Kubba & Hoomod, 2019).

Image encryption, specifically, is the process of securing image data to prevent unauthorized access and comprehension. One notable algorithm used in image encryption is Salsa20, a stream cipher

*name of corresponding author

This is anCreative Commons License This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License.

algorithm developed by Daniel J. Bernstein in 2005. Salsa20 offers a potent combination of security and efficiency for safeguarding continuous streams of data bits. In image encryption using the Salsa20 algorithm, the image is transformed into a stream of bits and then encrypted using a symmetric key. Notably, Salsa20 supports symmetric keys of up to 256 bits, providing a high level of security that can withstand brute-force attacks. The algorithm employs merging, repetition, and bit replacement operations to generate a random and unpredictable keystream, which is then XORed with the image bit stream to produce encrypted data (Fernandez de Loaysa Babiano et al., 2023; Mohaisen & Mohammed, 2020; Reza et al., 2020). The decryption process mirrors this procedure, utilizing the same key to restore the data to its original form.

Stream ciphers operate by encrypting and decrypting data bit by bit, akin to a flowing stream of data. Their primary advantage lies in their processing speed (Reza et al., 2020). By employing a stream cipher such as Salsa20, users can swiftly and efficiently secure their data.

The pycryptodome package is a prevalent choice in research, particularly for text encryption (Pabbi et al., 2021; Singhal* et al., 2020). Previously, research had been carried out with a similar theme, such as research conducted by Rasha Subhi Hameed, Bashar Ahmed Khalaf, and Ali Husein in 2020, in this research paper an enhanced iteration of the Salsa20 stream cipher is introduced, named ESalsa20. The primary objective is to boost the cipher's operational efficiency by incorporating two distinct chaotic maps, specifically the 1D logistic map and the Chebyshev map. ESalsa20 is meticulously crafted to cater to scenarios where the need for swift data encryption is as crucial as ensuring data security (Hameed et al., 2020). Through histogram analysis with Lena images at 256*256, evaluation of the results shows that the histogram is very randomly distributed, the MSE value is 6900, and the PSNR value is 9.74 dB. In research conducted by Eman L. Mohaisen and Rana Saad in 2020, they also proposed an image encryption method using the Salsa20 algorithm combined with chaotic maps, using lena images. This research produces a randomly distributed histogram and obtains an entropy value of 7.75, NPCR value of 97.34 %, UACI value of 32.23 %, MSE value of 8945, and PSNR 38.51 dB (Mohaisen & Mohammed, 2020).

Based on the preceding explanation, our research aimed to assess the viability and effectiveness of employing the Salsa20 algorithm for encrypting images using pycryptodome package with our proposed method. The Python version used in this research is 3.10.12 and the pycryptodome package version is 3.18.0.

METHOD

Data Collection

In The datasets used in this research include clegg.tif with a size of 815*880 pixels, lena.tif with a size of 512*512 pixels, monarch.tif with a size of 768*512, peppers.tif with a size of 512*512, and tulips. tif with a size of 768*512 pixels. The image dataset can be found at the following link https://links.uwaterloo.ca/Repository.html. The image dataset can be seen in Fig 1.

Fig. 1 Dataset

Salsa20 Algorithm

The Salsa20 algorithm is an encryption algorithm that is included in the 256-bit stream cipher category which was designed in 2005 (Fernandez de Loaysa Babiano et al., 2023; Mohaisen & Mohammed, 2020; Reza et al., 2020). Apart from supporting keys with a length of 256-bit, Salsa20 also

*name of corresponding author

This is anCreative Commons License This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License.

supports keys with a length of 128-bit. When using a 128-bit key, the key will be doubled to 256-bit. Salsa20 runs on 32-bit words. Salsa20's core functions change 256-bit keys (k_0, k_1, k_2, k_3, k_4, k_5, k_6, k_7), 64-bit counters (t_0, t_1), 64-bit nonce (v_0, v_1), and four 32-bit constants (c_0, c_1, c_2, c_3) make 512-bits. These inputs are mapped into a two-dimensional matrix as follows (1).

$$\begin{bmatrix} x_0 & x_1 & x_2 & x_3 \\ x_4 & x_5 & x_6 & x_7 \\ x_8 & x_9 & x_{10} & x_{11} \\ x_{12} & x_{13} & x_{14} & x_{15} \end{bmatrix} \leftarrow \begin{bmatrix} c_0 & k_0 & k_1 & k_2 \\ k_3 & c_1 & v_0 & v_1 \\ t_0 & t_1 & c_2 & k_4 \\ k_5 & k_6 & k_7 & c_3 \end{bmatrix} \quad (1)$$

The matrix in (1) will then be shuffled using the quarterround function depicted in Figure 2, the quarterround function involves the addition operation of modulus 2^{32} (+), XOR $\oplus$, and left-shift operation (<<<) on a 32-bit word input. This operation is carried out repeatedly to produce a keystream of sufficient length to encrypt or decrypt the data. To randomize it first use the columnround function by applying the quarterround function to each column of the matrix, after that the rowround function is applied which applies the quarterround function to each row of the matrix. The combination of these two functions is also called the doubleround function. After that, the littleendian function is applied to convert the data from the big-endian byte sequence to the little-endian byte sequence (Jawad Kubba & Hoomod, 2019; Muhalhal & Alshawi, 2022; Waleed et al., 2021). When finished, the keystream will be created using (2).

$$keystream = X + doubleround^{r}(x) \qquad (2)$$

Encryption is carried out by performing an XOR $\oplus$ operation between the keystream and the plaintext and decryption is carried out by performing an XOR $\oplus$ operation between the keystream and the ciphertext.

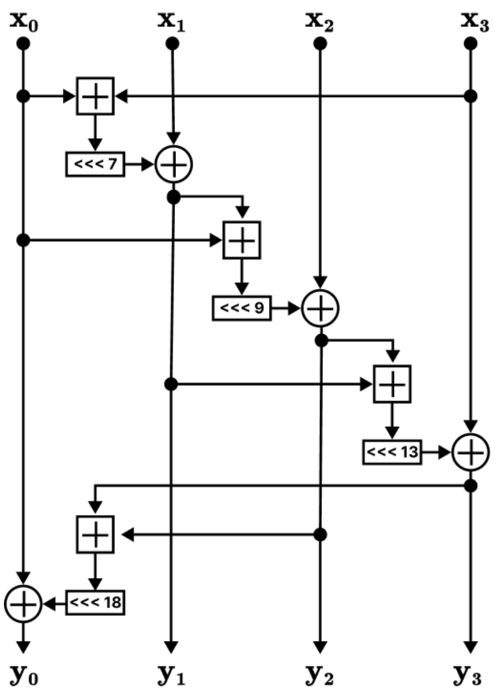


Fig. 2 Quarterround Function

Hash Function SHA-256

In short, a hash function is a function that converts an arbitrary input message into a message of a certain length according to the hash function used (M. R. Anwar et al., 2021) as shown in Fig 3. The SHA-256 hash function can calculate a 256-bit hash value for a 512-bit input message. If the input

*name of corresponding author

This is anCreative Commons License This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License.

message is more than 512-bit, the message is divided into many 512-bit data blocks. If the last block is smaller than 512-bit, then the block will be filled with one 1-bit until the block is 512-bit long. The SHA-256 algorithm calculates the hash value for each data block one by one, the hash result of the block becomes the input hash for calculating the hash of the next data block. The results of this last data block represent the hash value of the entire message (M. R. Anwar et al., 2021; Martino & Cilardo, 2020; Tran et al., 2021).

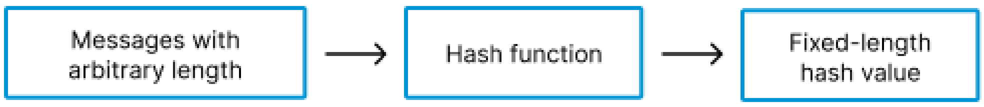


Fig. 3 Hash Function

Image Encryption and Decryption Method Proposed

The image is first read for each pixel, and then the values of the red, green, and blue colors are read as bytes, after that, the image bytes will be encrypted using a nonce consisting of 8 random bytes taken from a password from the word 'Password)(*&^%$#@!123' that converted using the SHA-256 hash function, so that whatever the length of the word, the key will remain consistent at 256-bit. The resulting ciphertext bytes will then be combined with a nonce which is then exported into an encrypted image.

For the decryption process, the encryption image will be read for each pixel, then read the values of the red, green, and blue colors as bytes, then take the 8-byte nonce at the beginning of the image byte and insert 8 bytes of the number 0 at the end of the image byte so that there is no error when exporting the image.

The method used to encrypt images can be seen in Figure 4. The PIL package is used to read images and export images.

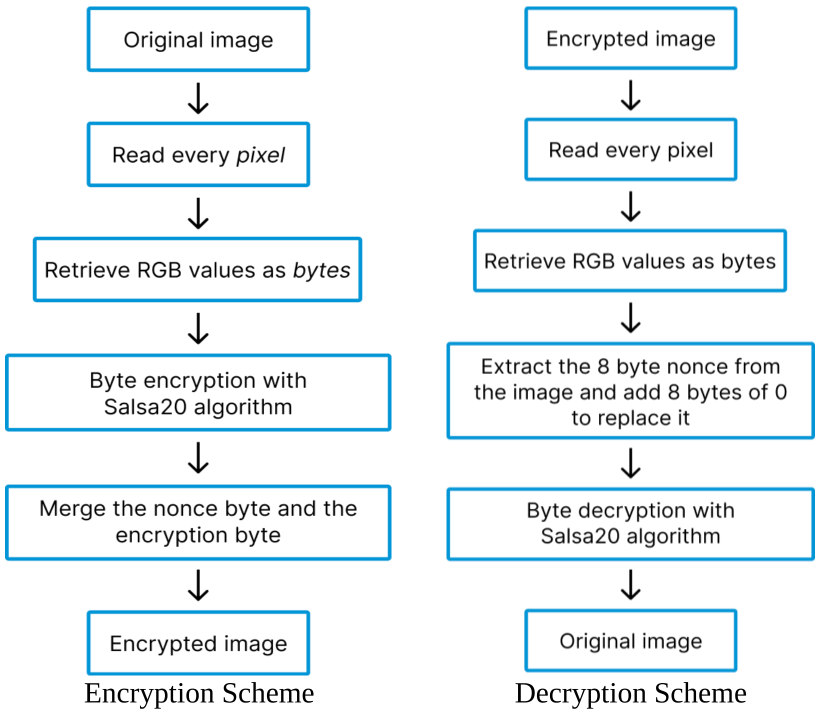

Original image
Read every pixel
Retrieve RGB values as bytes
Byte encryption with Salsa20 algorithm
Merge the nonce byte and the encryption byte
Encrypted image
Encryption Scheme
Encrypted image
Read every pixel
Retrieve RGB values as bytes
Extract the 8 byte nonce from the image and add 8 bytes of 0 to replace it
Byte decryption with Salsa20 algorithm
Original image
Decryption Scheme

Fig. 4 Proposed Encryption and Decryption method

The longer the possible key, the more resistant it will be to brute-force attacks. In this study, the key length used was 256-bit. The number of possible keys in the 256-bit key space is 2^{256}. This is a very large number and is difficult to guess randomly through brute-force attacks. Salsa20 security is also

*name of corresponding author

This is anCreative Commons License This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License.

influenced by other factors such as the strength of the hash function, the use of random nonce, and the overall implementation of the algorithm.

Evaluation of the Quality of Encryption and Decryption Results

Histogram analysis can be done by comparing the histogram of the original image with the histogram of the image that has undergone the encryption process. By examining the distribution of pixel intensities in an image to obtain information about the distribution of pixel values contained in it, we can identify patterns, symmetry, brightness and contrast in the image which can help us understand its visual characteristics (Arab et al., 2019; Gupta et al., 2020; Jangir & Pandey, 2021; Mansouri & Wang, 2021), for calculations, 256 bins are used and the image will be converted to black and white before testing.

Due to evaluate the effectiveness of encryption, we also calculate entropy. Entropy is a matrix that provides information about the complexity of the patterns contained in the distribution of pixel values in the image. To simplify calculations, a probability distribution from the histogram is used which is then applied to the entropy formula from Shannon Entropy (SE) (Cincotta et al., 2021; Mansouri & Wang, 2021; Naif et al., 2023; Zou et al., 2020). Equation (3) explains how to calculate Shannon Entropy. If the encryption succeeds in randomizing the data well, then the entropy value of the encrypted image will be high, approaching 8, which indicates that the information in the image has been randomly distorted and is difficult to predict.

$$H(P_i) = \sum_{l-1}^{L} P(l)\, log(P(l)) \qquad (3)$$

The values ($P_1, P_2, P_3, \ldots, P_k$) are blocks obtained from image P and $H(P_i)$ is (SE) of that block where $P(l)$is the probability of l entropy value calculated for the original image and the resulting image encryption is presented in Table 1. Two essential tests for security analysis against differential attacks are the Number of Pixel Change Rate (NPCR) and Unified Average Changing Intensity (UACI). NPCR measures the rate of change in pixel values in an encrypted image when there is a random modification of one pixel in the original image. For example, I_1 and I_2 are two different images with difference of 1 pixel, and E_1 and E_2 are encrypted images.

$$NPCR = \frac{\sum_{i=1}^{M}\sum_{j=1}^{N} D(i,j)}{M\ x\ N} x\ 100\% \qquad (4)$$

The NPCR of the encrypted image is calculated using the formula in (4), where N and M are the width and height of the image.

$$D(i,j) = \begin{cases} 0, when\ E_1(i,j) = E_2(i,j) \\ 1, when\ E_1(i,j) \neq E_2(i,j) \end{cases} \qquad (5)$$

D(i, j) is a bipolar row of the same size as E_1 and E_2 (S. Anwar & Meghana, 2019; Jangir & Pandey, 2021; Mansouri & Wang, 2021) as given by (5).

$$UACI = \sum_{i=1}^{M} \sum_{j=1}^{N} \left(\frac{E_1(i,j) - E_2(i,j)}{W\ x\ H}\right) x\ 100\% \qquad (6)$$

Here, E(i, j) is the pixel value at position (i, j) of the encrypted image. UACI is used to measure the average intensity of the encrypted image which can be calculated using (6). In UACI and NPCR calculations the image is converted to black and white to focus on image structure, color consistency, and lighten computation (Jangir & Pandey, 2021; Mansouri & Wang, 2021).

Peak Signal-to-Noise Ratio (PSNR) is used to measure the level of distortion or noise that appears during the encryption and decryption process. Meanwhile, Mean Square Error (MSE) is used to compare pixel values between the original image and the encrypted image (S. Anwar & Meghana, 2019; Jangir & Pandey, 2021; Setiadi, 2021). MSE can be calculated with (7).

*name of corresponding author

This is anCreative Commons License This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License.

$$MSE(f,g) = \frac{1}{mn}\sum_{i=1}^{m} \sum_{j=1}^{n}(f_{ij} - g_{ij})^2 \quad (7)$$

Where f = data matrix from the original image, g = Data matrix from the encrypted image, m = Pixel row number with row index i, n = Pixel column number with column index j. PSNR and MSE are inversely related, where PSNR will decrease as the MSE value increases. If the PSNR value is above 30 dB then it can be said to be ideal (Setiadi, 2021).

$$PSNR = 10log_{10}\left(\frac{maxval^2}{MSE}\right) \quad (8)$$

Equation (8) shows how to calculare PSNR, maxval value is the maximum signal value in the original image. PSNR provides an indication of the extent to which the decrypted image approaches the original image in terms of quality. The higher the PSNR value, the better the decryption quality.

RESULT

After evaluating using 5 images, Clegg, Lena, Monarch, Peppers, and Tulips, we get the following results, in Table 1 show that the NPCR value produced is more than 99% and the average UACI value produced is more than 50%. This NPCR value shows that the method above is very effective in randomizing image pixels into an encrypted image. This is supported by the relatively high UACI value, indicating that the change in pixel intensity in the encrypted image is high. Apart from that, the MSE value of each image is also close to 0, which indicates that there is only a small error in the decrypted image, then the average PSNR value obtained is more than 50 dB. This is shown a small error rate, the PSNR value is relatively high and indicates that the image quality is ideal after the decryption process.

Table 1. Entropy, MSE, PSNR, NPCR, and UACI Calculation Results

Image	*Size*	*Entropy*		*NPCR*	*UACI*	*MSE*	*PSNR*
		Original	*Encrypted*				
clegg	815 x 880	*7,67*	*7,63*	99,56	69,28	0,08	58,62
lena	512 x 512	*7,44*	*7,63*	99,45	55,87	0,09	58,45
monarch	768 x 512	*7,18*	*7,62*	99,46	55,14	0,04	61,89
peppers	512 x 512	*7,57*	*7,63*	99,46	59,85	0,03	52,84
tulips	768 x 512	*7,69*	*7,63*	99,53	66,68	0,10	57,71

Here, we got similar entropy values indicating that the encryption algorithm performs consistently across images. This suggests that the algorithm has no bias towards any type of images and likely produces a similar degree of randomness. An entropy value close to 8 also means that the information in the image is random and difficult to predict.

DISCUSSIONS

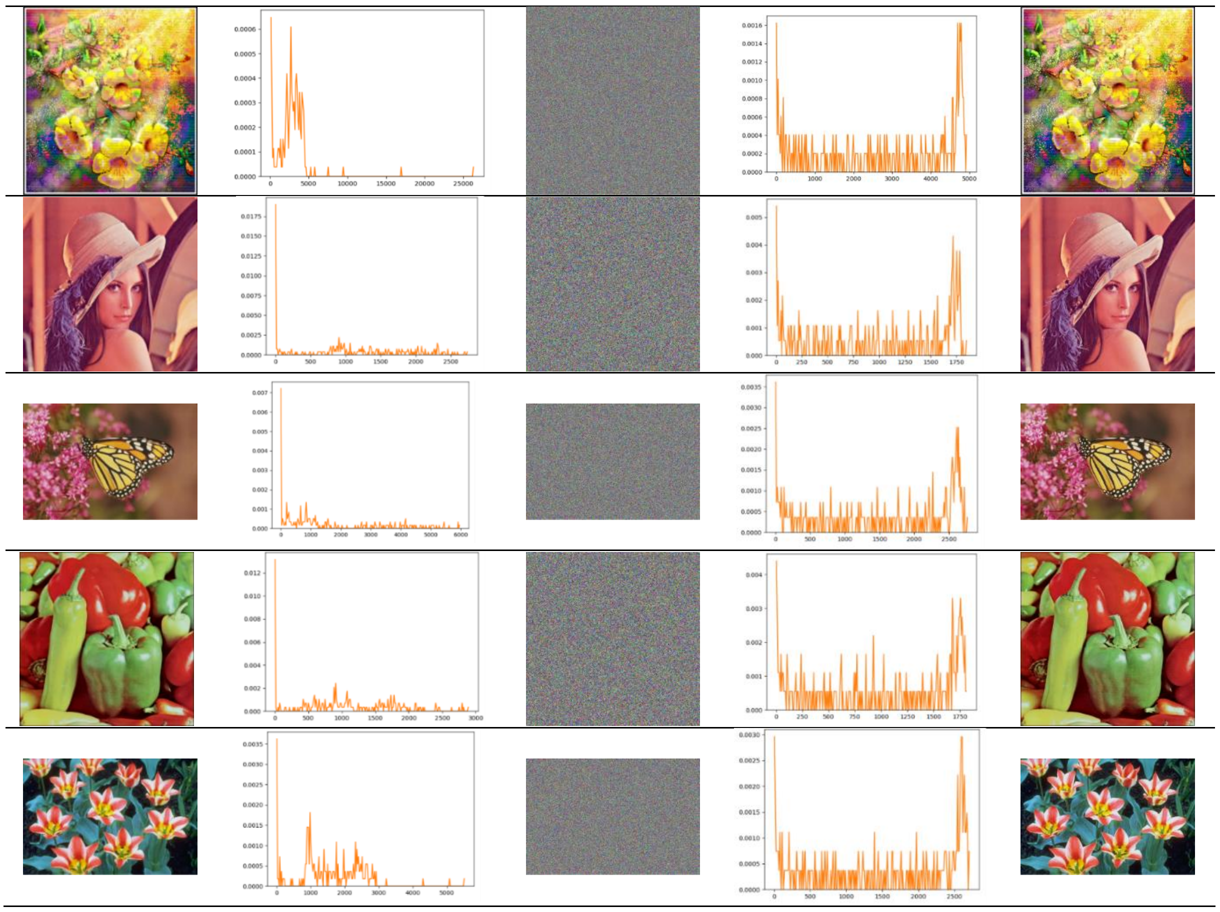
Through this research, We try to encrypt images by reading each pixel in the image and converting the red, green, and blue values in it to bytes so that the image data obtained can be encrypted or decrypted using the *pycryptodome* package. After evaluation, the results we got were quite satisfactory, but the encrypted image showed a histogram that was not evenly distributed. In Table 2 the encrypted image histogram shows that there are parts of the histogram that are not evenly distributed, this is because not all of the ciphertext bytes are ciphertext, in the first 8 bytes of data a nonce is inserted for the decryption process, while the rest which is evenly distributed is the actual ciphertext. The entropy calculation results in Table 1 show that the image encryption results are consistent at 7.62 and 7.63 for all images even though the original images have varying entropy.

Table 2. Comparison Result of Image and Histogram

Original	Original Histogram	Encrypted	Encrypted Histogram	Decrypted

*name of corresponding author

This is anCreative Commons License This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License.



Based on Table 1 we got quite ideal values, this is because the image differences are only located in the last 8 pixels, this is caused by the addition of 8 nonce bytes which are inserted at the beginning of the ciphertext byte before it is exported into an image, the last 8 ciphertext bytes will be lost in image export process because the pixel bytes have been fulfilled according to the size of the original image. Because the initial 8 bytes have been taken for the nonce, in the decryption process 8 bytes of data need to be added to complete the missing bytes, in this study I inserted 8 bytes 0 to complete it, this causes the last 3 pixels to be an error because each byte consists of 3 bytes, namely red, green, and blue as shown in Fig 5. An NPCR value close to 100% explains that only a slight pixel difference occurs. The difference in the image will be visible if the decrypted image is enlarged, the last 3 pixels will be visible which are different from the original image as shown in Fig 5.

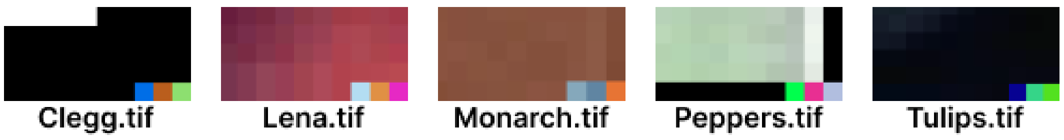


Fig. 5 Pixel Error

Based on previous research, image encryption was carried out with Salsa20 with results of MSE 8945, PSNR -39.51, NPCR 97.34, and UACI 32.23 on Lena Image at size 256*256 (Mohaisen & Mohammed, 2020). This research obtained higher results than previous research, namely MSE 0.09, PSNR 58.45, NPCR 99.45, UACI 55.87, despite using the same image at a higher resolution of 512*512.

CONCLUSION

In this research, image encryption was carried out using pycryptodome package with the Salsa20 algorithm. The key used is 256-bit long which is obtained from key conversion using SHA-256, even

This is anCreative Commons License This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License.

e-ISSN : 2541-2019
p-ISSN : 2541-044X

though the attacker knows how to get the nonce from the image, the key space has a combination of 2^{256} in length so it is quite safe from brute force attacks. Histogram analysis shows that the histogram is not distributed evenly due to the addition of 8 bytes inserted at the beginning of the ciphertext byte. The entropy value shows that the encrypted image is consistent at 7.62 and 7.63 even though the original image has various entropy. The NPCR value produced is more than 99% and the highest UACI value produced is 69.28%, based on these results it has been seen that the encryption results are effective in randomizing image pixels, this is supported by the high UACI value, which means that the change in pixel intensity in the encryption image is relatively high. The MSE value obtained is close to 0, which means very few errors occur, apart from that the resulting PSNR value is more than 50 dB, this value can be categorized as ideal. Based on the data above we can conclude that our proposed method is effective and feasible for image encryption with the Salsa20 algorithm, to get the same image decryption results as the original image can be done by generating a nonce without inserting it in the image.

REFERENCES

Anwar, M. R., Apriani, D., & Adianita, I. R. (2021). Hash Algorithm in Verification Of Certificate Data Integrity And Security. *Aptisi Transactions on Technopreneurship (ATT)*, *3*(2), 65–72. https://doi.org/10.34306/att.v3i2.212

Anwar, S., & Meghana, S. (2019). A pixel permutation-based image encryption technique using chaotic map. *Multimedia Tools and Applications*, *78*(19), 27569–27590. https://doi.org/10.1007/s11042-019-07852-2

Arab, A., Rostami, M. J., & Ghavami, B. (2019). An image encryption method based on chaos system and AES algorithm. *Journal of Supercomputing*, *75*(10), 6663–6682. https://doi.org/10.1007/s11227-019-02878-7

Cincotta, P. M., Giordano, C. M., Alves Silva, R., & Beaugé, C. (2021). The Shannon entropy: An efficient indicator of dynamical stability. *Physica D: Nonlinear Phenomena*, *417*. https://doi.org/10.1016/j.physd.2020.132816

Fernandez de Loaysa Babiano, L., Macfarlane, R., & Davies, S. R. (2023). Evaluation of live forensic techniques, towards Salsa20-Based cryptographic ransomware mitigation. *Forensic Science International: Digital Investigation*, *46*. https://doi.org/10.1016/j.fsidi.2023.301572

Gupta, A., Singh, D., & Kaur, M. (2020). An efficient image encryption using non-dominated sorting genetic algorithm-III based 4-D chaotic maps: Image encryption. *Journal of Ambient Intelligence and Humanized Computing*, *11*(3), 1309–1324. https://doi.org/10.1007/s12652-019-01493-x

Hameed, R. S., Hussein, A., Khalaf, B. A., Fadel, A. H., Hasoon, J. N., Mostafa, S. A., & Khalaf, A. (2020). A Light-weight ESalsa20 Ciphering based on 1D Logistic and Chebyshev Chaotic Maps. *Solid State Technology*. https://www.researchgate.net/publication/344492787

Jangir, A., & Pandey, J. G. (2021). GIFT cipher usage in image data security: hardware implementations, performance and statistical analyses. *Journal of Real-Time Image Processing*, *18*(6), 2551–2567. https://doi.org/10.1007/s11554-021-01146-3

Jawad Kubba, Z. M., & Hoomod, H. K. (2019). A Hybrid Modified Lightweight Algorithm Combined of Two Cryptography Algorithms PRESENT and Salsa20 Using Chaotic System. *International Conference of Computer and Applied Sciences*, 199–203.

Mansouri, A., & Wang, X. (2021). Image encryption using shuffled Arnold map and multiple values manipulations. *Visual Computer*, *37*(1), 189–200. https://doi.org/10.1007/s00371-020-01791-y

Martino, R., & Cilardo, A. (2020). Designing a SHA-256 processor for blockchain-based IoT applications. *Internet of Things*, *11*. https://doi.org/10.1016/j.iot.2020.100254

Mohaisen, E. L., & Mohammed, R. S. (2020). Improving Salsa20 stream cipher using random chaotic maps. *2020 3rd International Conference on Engineering Technology and Its Applications, IICETA 2020*, 1–6. https://doi.org/10.1109/IICETA50496.2020.9318902

Muhalhal, L. A., & Alshawi, I. S. (2022). Improved Salsa20 Stream Cipher Diffusion Based on Random Chaotic Maps. *Informatica (Slovenia)*, *46*(7), 95–102. https://doi.org/10.31449/inf.v46i7.4279

Naif, J. R., Ahmed, I. S., Alani, N., & Hoomod, H. K. (2023). EAMSA 512: New 512 Bits Encryption Al-gorithm Based on Modified SALSA20. *Iraqi Journal for Computer Science and Mathematics*, 131–142. https://doi.org/10.52866/ijcsm.2023.02.02.011

This is anCreative Commons License This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License.

Pabbi, A., Malhotra, R., & Manikandan, K. (2021). Implementation of least significant bit image steganography with advanced encryption standard. *2021 International Conference on Emerging Smart Computing and Informatics, ESCI 2021*, 363–366. https://doi.org/10.1109/ESCI50559.2021.9396884

Reza, S. M. S., Arifeen, M. M., Tiong, S. K., Akhteruzzaman, M., Amin, N., Shakeri, M., Ayob, A., & Hussain, A. (2020). Salsa20 based lightweight security scheme for smart meter communication in smart grid. *Telkomnika (Telecommunication Computing Electronics and Control)*, *18*(1), 228–233. https://doi.org/10.12928/TELKOMNIKA.V18I1.14798

Setiadi, D. R. I. M. (2021). PSNR vs SSIM: imperceptibility quality assessment for image steganography. *Multimedia Tools and Applications*, *80*(6), 8423–8444. https://doi.org/10.1007/s11042-020-10035-z

Singhal*, Mr. V., Shukla, Mr. Y. K., & Prakash, Dr. N. (2020). Image Steganography embedded with Advance Encryption Standard (AES) securing with SHA-256. *International Journal of Innovative Technology and Exploring Engineering*, *9*(8), 641–648. https://doi.org/10.35940/ijitee.H6442.069820

Tran, T. H., Pham, H. L., & Nakashima, Y. (2021). A High-Performance Multimem SHA-256 Accelerator for Society 5.0. *IEEE Access*, *9*, 39182–39192. https://doi.org/10.1109/ACCESS.2021.3063485

Waleed, J., Noori Mazher, A., & Tariq MaoLood, A. (2021). Developed Lightweight Cryptographic Algorithms for The Application of Image Encryption: A Review. *Journal of Al-Qadisiyah for Computer Science and Mathematics*, *13*(2), 11–22. https://doi.org/10.29304/jqcm.2021.13.2.788

Zou, Y., Zhang, J., Upadhyay, M., Sun, S., & Jiang, T. (2020). Automatic image thresholding based on Shannon entropy difference and dynamic synergic entropy. *IEEE Access*, *8*, 171218–171239. https://doi.org/10.1109/ACCESS.2020.3024718

This is anCreative Commons License This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License.